

MEMORANDUM

RM-3588-PR

MAY 1963

THE HEURISTIC COMPILER

Herbert A. Simon

PREPARED FOR:

UNITED STATES AIR FORCE PROJECT RAND

The **RAND** *Corporation*
SANTA MONICA • CALIFORNIA

MEMORANDUM

RM-3588-PR

MAY 1963

THE HEURISTIC COMPILER

Herbert A. Simon

This research is sponsored by the United States Air Force under Project RAND—contract No. AF 49(638)-700 monitored by the Directorate of Development Planning, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

The **RAND** *Corporation*

1700 MAIN ST • SANTA MONICA • CALIFORNIA

Copyright © 1963

THE RAND CORPORATION

PREFACE

The research reported in this Memorandum, rather than being aimed at the construction of a specific compiler (a computer program for translating instructions from a language convenient to the programmer to the machine language), is directed towards deepening our understanding of the kinds of problem-solving activity that are involved in computer programming, and the kinds of language and representational means that are needed to produce more sophisticated compilers. The Memorandum takes the form, therefore, of a series of illustrative problems of compiler design, with proposals, worked out in some detail, for their solution.

There has been, in the past decade, enormous progress in the development of higher-level programming languages for instructing computers. Through the invention of algebraic compilers, like FORTRAN, IT, and ALGOL, data processing languages like COBOL, and list-processing languages like IPL, LISP, and COMIT, the labor of programming has been reduced several orders of magnitude. Yet, when we are faced with the complexities of modern command and control systems, and the programming problems they present, we recognize that the progress made to date is not nearly enough. Programming a computer to perform a complex task is still very much more intricate and tedious than instructing an intelligent and trained human being for that task.

Most visibly, the human--if, to repeat, he is intelligent and knowledgeable--does not have the literalness of mind that is so characteristic of the computer, and so exasperating in our interaction with it. From his own store of knowledge, he supplies facts that we neglect to give him; given statements of objectives in broad functional terms, he applies his problem-solving powers to filling in the detail of method; confronted with the vagueness and informality of natural language, he interprets meaning and intent.

The experiments reported in this Memorandum aim at further bridging the gap between the explicitness of existing computer programming languages and the freedom and flexibility of human communication. The work was motivated by the belief that until we bridge that gap, we shall not be able to harness effectively the powers made available to us by modern digital computers and apply them to the vast systems (e.g., command and control) that are becoming such a central feature of our military and civilian technology.

This work is part of The RAND Corporation's continuing program of research in the area of complex information processing, under U.S. Air Force Project RAND. The author, a RAND consultant, is a member of the faculty at Carnegie Institute of Technology. The Memorandum is directed primarily to systems programmers who are faced with problems

of compiler design, with the hope that its proposals will suggest means for increasing the power, generality, and flexibility of compiling systems.

SUMMARY

Two major themes run through the topics discussed in this Memorandum. The first thesis is that more of the programming burden can be shifted from programmer to computer if the computer is given some problem-solving powers. In previous works it has been shown how a computer program, the General Problem-Solver, can simulate the kinds of means-end analysis that humans use to solve problems.* Part I of the present Memorandum shows how a compiler can be designed that makes use of heuristic problem-solving techniques like those incorporated in the General Problem-Solver (GPS). Such a scheme permits a desired program to be specified in general terms, with the compiler using means-end analysis and selective trial-and-error search to work out the exact "how" of it.

The second main theme is that if we are to have flexibility in a compiler language commensurate with the flexibility of natural language, we must first gain an

*The General Problem-Solver has been described, and its behavior analyzed, in several RAND publications by A. Newell, J. C. Shaw, and H. A. Simon: Report on a General Problem-Solving Program, P-1584, February 9, 1959; The Simulation of Human Thought, P-1734, June 22, 1959; A Variety of Intelligent Learning in a General Problem Solver, P-1742, July 6, 1959; GPS: A Program that Simulates Human Thought, P-2257, April 10, 1961; Computer Simulation of Human Thinking, P-2276, April 20, 1961; Computer Simulation of Human Thinking and Problem Solving, P-2312, May 29, 1961.

Also see, A Guide to the General Problem-Solver Program GPS-2-2, Allen Newell, RM-3337-PR, February 1963.

understanding of the ways in which meanings are represented in natural language, and then devise representations of corresponding power (and ambiguity) for compiling languages. Parts II and III are devoted primarily to questions of language and representation. They provide a number of suggestions for increasing the generality and flexibility of compiler languages.

The boundaries between the three parts are largely chronological. Part I represents work completed during the winter of 1960-61; Part II, work done during the spring of 1961; and Part III, work done since the summer of 1961, particularly during the summer and autumn of 1962. A program listing for the main portion of the compiling scheme, described in Sections I-V of Part I, is given in Appendix A. A program listing for the annexing scheme, described in Sec. IX of Part III, is given in Appendix B.

ACKNOWLEDGMENT

I am greatly indebted to H. S. Kelly, of The RAND Corporation, for numerous discussions of the work described in this paper, and for help in solving many of the problems that arose.

CONTENTS

PREFACE	iii
SUMMARY	vii
ACKNOWLEDGMENT	ix
<u>PART I: A PROBLEM-SOLVING COMPILER</u>	1
I. THEORY OF PROBLEM SOLVING	2
II. PROGRAM WRITING AS PROBLEM SOLVING	5
III. OUTLINE OF A HEURISTIC CODER FOR IPL-V ...	6
The SDSC Compiler	7
The DSCN Compiler	12
The General Compiler	15
IV. RELATION OF THE HEURISTIC COMPILER TO GPS.	20
V. FLOW DIAGRAMS	21
<u>PART II: GENERAL IMPLICATIONS FOR REPRESENTATIONS.</u>	27
VI. LANGUAGE AND REPRESENTATIONS IN THE COMPILER	28
Descriptive Names	28
State Descriptions	31
VII. THE DESIGN OF REPRESENTATIONS	34
<u>PART III: EXPERIMENTS WITH REPRESENTATIONS</u>	41
VIII. GENERALIZED PROCESSES	44
"FIND" Processes	44
"SORT" Processes	48
Recursive Functional Languages	51
IX. DEFINITE DESCRIPTIONS	53
Syntactic Characterization	53
Semantic Characterization	54
Annexing Descriptive Information to an Information Store	55
X. COMPILATION OF ROUTINES FROM PARTIAL DESCRIPTIONS	60

Appendix

A.	PROGRAM LISTING OF THE HEURISTIC COMPILER	65
B.	PROGRAM LISTING OF AN INFORMATION- ANNEXING SCHEME	118

PART I

A PROBLEM-SOLVING COMPILER

In this first part, we describe a compiler that makes use of heuristic problem-solving techniques like those incorporated in the General Problem-Solver (GPS).^{*} Section I provides a brief introduction to the theory of problem solving and the structure of GPS. Section II shows how programming can be interpreted as a problem-solving activity, within this framework. Section III describes the main components of the compiler, and Sec. IV indicates how the compiler could be incorporated as a set of subroutines to GPS. Section V describes an extension of the compiler to handle flow diagrams. A complete program listing for the compiler is given in Appendix A.

^{*}Op. cit., p. vii.

I. THEORY OF PROBLEM SOLVING

The motivation for the Heuristic Compiler is supplied by a theory of problem solving that also provides the basic framework for the General Problem-Solver. By a problem, we mean a situation of the following kind:

1. We are given a (partial) description of a present situation and a desired situation. These situations are described in a language that we may call the state language. The state language is sufficiently rich to permit us to describe situations (we shall call such descriptions objects), and to describe differences between pairs of situations.
2. We are given a list of operators that can be applied to situations to transform them into new situations. Operators are named in a language that we may call the process language. Any sequence of operators named in the process language, also names an operator--the compound operator that consists in applying, successively, the elementary operators belonging to the sequence.
3. A problem solution is a (compound) operator in the process language that will transform the object describing the present situation into the object describing the desired situation.

EXAMPLE: We take as the objects in the state language the integers, 1, 2, We take as the elementary operator the successor operation, which we shall designate as ' in the process language. Then '' and '''' are examples of compound operators. Consider the problem of transforming the present object 5 into the goal object 8. The solution is the operator '', for $5'' = 8$. More generally, '' is the operator that removes the difference +3 between any two objects, x and y; for if $y - x = +3$, then $x'' = y$. Here +3 is a difference in the state language; '' is the operator relevant to that difference in the process language. We may construct a table of connections to associate with each difference the operator or operators relevant to it.

With this formulation of the concept of "problem," many techniques of problem solving can be subsumed under the following general paradigm:

MEANS-END ANALYSIS. Given the present and desired objects, find a difference between them. Next find an operator relevant to the difference; determine if the operator can be applied to the present object. If so, apply it. (If not, describe the objects to which it would apply and transform the present object into an object of that kind--a new "desired object.") Take the new object thus obtained as the present object and repeat the process.

The General Problem-Solver is a program that uses this scheme of means-end analysis for attempting the solution of any problem that can be cast into the form described.*

*This is a bare-bone description of GPS, but it will suffice for our purposes.

II. PROGRAM WRITING AS PROBLEM SOLVING

The task of proving a theorem can be formulated as a problem for GPS. The desired object is the theorem to be proved. The present object is the set of axioms and already-proved theorems. The operators are the legitimate processes for transforming a subset of axioms and/or theorems into a new theorem. We have a proof when we have a sequence of operators that transforms the present object into the desired object. (What we call a proof here is usually regarded as the justification for the proof steps; the proof, as usually written out, consists of the sequence of successive transformations of the axioms and given theorems.)

The sequence of operators that constitutes a proof can also be interpreted as a program that generates the desired object from the given object; for if we apply the operators of the proof, in sequence, to the present object (the axioms and previous theorems), we obtain precisely the desired object--the theorem to be proved. Thus, a theorem-proving system can be regarded, at least formally, as a program-writing system. Conversely, if we can formulate a programming goal as a difference between a present and a desired object, we can presumably use the same processes, which in the other context will generate the proof of a theorem, to generate a program.

III. OUTLINE OF A HEURISTIC CODER FOR IPL-V

In the remainder of Part I, I shall describe a number of routines for compiling programs in Information Processing Language V (IPL-V), an interpretive list-processing language.* What is common to all of these compiling procedures is that they embody the problem-solving notions discussed in the preceding paragraphs. That is, each of the compiling routines accepts the task of writing programs in IPL-V on the basis of certain information provided to it. The task is accomplished by the application of the means-end analysis that has been described. The several compiling routines differ with respect to their methods of formulating or representing the problem--that is, each operates with a different state language. At present, there are three compiling routines:

1. SDSC Compiler (State Description Compiler) [U140].**

This routine takes as its input a description (SDSC) of the contents of the relevant computer cells before and after the routine to be compiled has been executed. It produces an IPL-V routine that will transform the input state description into the output state description.

* See Newell, Allen (Ed.), Information Processing Language V Manual, Prentice-Hall, Englewood Cliffs, N.J., 1961.

**Expressions in square brackets are names of the corresponding routines, data list structures, and symbols in the compiler program.

2. DSCN Compiler (Descriptive Name Compiler) [U134].

This routine takes as its input a verbal definition (in the form of an imperative sentence) of the routine to be compiled. It produces an IPL-V routine that is the translation, in the interpretive language, of that definition.

3. General Compiler [U135]. This is an executive routine that can use the SDSC Compiler, the DSCN Compiler, and others as subroutines. It takes as its input information about the routine to be compiled; the information can be stated in any one of several representations (e.g., those appropriate to SDSC or DSCN). The routine then selects subroutines that can use this information to produce the desired IPL-V code.

From a logical standpoint, we could describe the Heuristic Coder as a single program whose executive routine is the General Compiler, and which contains the SDSC Compiler and the DSCN Compiler as subroutines. For clarity of exposition, it will be better to describe the two parts first as independent programs, and then show how they are imbedded in the General Compiler.

THE SDSC COMPILER

A computer routine can be defined by specifying the changes it produces in the contents of the storage location it affects, or, what amounts to almost the same thing, by specifying the before and after conditions of

these storage registers. A definition of this kind is not, of course, univocal, for programming is a synthetic, not an analytic task; there will generally be many programs (not all equally efficient or elegant) that will do the same work. As presently constituted, the SDSC Compiler attempts to find some one routine to accomplish a given task.

EXAMPLE: In IPL-V there is a process, PUT SYMBOL J3 IN MEMORY LOCATION H5, which affects a single memory location, H5. This process [X105] happens to have the name J3; its state description (SDSC) is the following:

BEFORE J3 is executed, cell H5 contains a symbol, call it S1, followed by an indeterminate list of symbols, R0 (call this the pushdown list associated with H5).

AFTER J3 has been executed, cell H5 contains the symbol J3, followed by the same list of symbols, R0, as before. The token of symbol S1 that was previously in H5 has been destroyed.

Notice that it is implicit in this SDSC definition of J3 that the content of no cell other than H5 has been altered by the routine. We can represent the SDSC [the value of attribute X24 of the routine] diagrammatically as follows:

SDSC of J3

Affected Cells

H5

Input
Output

S1,R0
J3,R0.

Generalizing, the SDSC of a routine consists of a list of affected cells [attribute X71 of X24]. For each affected cell on the list, the SDSC specifies its input state [attribute X75 of X71] and its output state [attribute X76 of X71].

To compile the IPL-V code (JDEF) for J3, the SDSC Compiler [U140] proceeds as follows:

1. It matches [X7] the input states with the output states of the affected cells until it finds a difference. In the example cited, the difference between the input and output states of H5 may be called a replacement in H5.
2. It searches [U150] a table of connections [X90] that associates with each difference a list of operators (compiled IPL-V routines) that are relevant to that difference. In the example, the table of connections would contain, associated with the replacement difference [X83], the IPL-V routine P2(C) [X106]. P2(C) replaces the symbol in cell C, a variable, with the symbol in cell H0. Thus, P2(C) has the following SDSC [see local 95 of X106]:

SDSC of P2(C)

	<u>Affected Cells</u>	
	<u>H0</u>	<u>C</u>
Input	S2,R0	S1,R0
Output	R0	S2,R0.

3. It tentatively applies [U141] the relevant operator it has found to the input state of the SDSC to be compiled, and determines the resulting output state. In applying the operator, it makes appropriate substitutions for the variables in the operator [U153, U154]. Thus, applying P2(C) to the input of J3, we find, by matching, that we should set C = H5 and S2 = J3, giving:

SDSC of P2(H5)

	<u>H0</u>	<u>H5</u>
Input	J3,R0	S1,R0
Output	R0	J3,R0.

4. The application of the operator creates two new subproblems: Let I_a be the input state of the routine to be compiled, O_a its output state, I_b the input state of the operator, and O_b its output state. The original problem was to transform I_a into O_a . The new problems are: (1) to transform I_a into I_b (i.e., to establish the input conditions for application of the operator), and (2) to transform O_b into O_a (i.e., to transform the output state of the operator into the desired output state of the routine to be compiled). Either of these new

problems may reduce to the identity transformation, in which case that part of the problem is solved. If this reduction does not occur, then the same steps, 1, 2, 3, are applied [recursion of U140] to the new subproblem.

In the example at hand, 0_b is identical with 0_a ; hence, the remaining subproblem is to transform I_a into I_b , that is, to compile a routine with SDSC:

H0
S2,R0
J3,S2,R0.

The repetition of step 1 for this subproblem discovers a new difference, an addition [X82] to H0. Step 2 finds the relevant operator, process P1(S) [X107], which adds the symbol S to the symbol list in H0. Applying, in step 3, the operator P1(J3), the input state of H0: S2,R0, is transformed into the output state, J3,S2,R0. Hence, the solution to the original problem of compiling J3 is obtained by the sequence, P1(J3), P2(H5), or, in the usual IPL-V format [X22 of the routine]:

J3 10 J3
 20 H5 0.

We see that for the SDSC compiler to operate, it must be provided with a set of differences and matching tests for noticing differences, a set of already-compiled operators, and a table of connections between differences and operators. Further, when it has compiled a new routine,

the compiler can annex this routine to its set of available operators and use it in compiling subsequent routines.

THE DSCN COMPILER

Let us now consider an alternative compiling scheme for the same routine, J3. Instead of specifying the before and after condition of the computer cells, we define the routine [X20 of the routine] in terms of the function it performs: REPLACE THE SYMBOL IN H5 BY J3. (This definition (DSCN) resembles more closely than the previous one the manner in which routines are defined for "conventional" compilers like FORTRAN or LISP. What distinguishes the present scheme from these is the use of heuristic means-end analysis for working from the definition to the compiled routine.)

The first step in the DSCN Compiler [U134] is to search a list of available (compiled) routines to find one whose DSCN is as similar as possible to the DSCN of the routine to be compiled. In the case at hand, we would find the routine P2(C): REPLACE THE SYMBOL IN C BY (H0).

At the second step [U130], means-end analysis is performed to transform the compiled routine that has been found into the new routine. The transformations are performed on the DSCN's. Thus, in the present example, there are two differences between P2(C) and J3. The

former refers to the cell C, the latter to H5; the former refers to the symbol that is contained in cell H0, the latter to the symbol J3.

The compiler notices these differences (in a sequence), and searches for an operator relevant to removing the differences. In this case, C can be transformed to H5 by a substitution operator. (H0) can be changed to J3 by an addition operator (MAKE (H0)=J3 BY ADDITION). The application of these operators to the DSCN of P2(C) would compile the desired routine in the following stages:

P2(C)	REPLACE THE SYMBOL IN C BY (H0).
Apply <u>substitution</u>	REPLACE THE SYMBOL IN H5 BY (H0).
Apply <u>addition</u>	REPLACE THE SYMBOL IN H5 BY J3.

The resulting program in this case is identical with that obtained by the SDSC Compiler.

A somewhat more complex routine compiled by the DSCN Compiler is:

J13: INSERT (1) AT THE END OF (THE VALUE OF ATTRIBUTE
(0) OF (2)).

The list of available IPL routines includes:

J65: INSERT (0) AT THE END OF (1).

The differences between J65 and J13 are in their arguments. J65 has the argument (0), where J13 has the argument (1); J65 has the argument (1), where J13 has the argument (THE VALUE OF ATTRIBUTE (0) OF (2)). Since

it is not easy in IPL-V to rearrange arguments that are located in the pushdown list of the communication cell, H0, the compiler facilitates matters by incorporating in the compiled routine an algorithm that moves the inputs of the routine to be compiled into known working storage locations, then puts these inputs back into H0 in the order in which they are needed for the subprocesses. That is, the compiler first transforms J13 into another routine, call it K13, which it then compiles. The DSCN of K13 is:

```
K13:  INSERT LW1 AT THE END OF THE VALUE OF  
      ATTRIBUTE LW0 OF LW2.
```

The code for J13 may be written as:

```
J13      J52  
          K13  
          J32      0 .
```

Now K13 is to be compiled with the aid of J65. Comparing the corresponding arguments of the two routines, we see that this involves finding the value of attribute LW0 of LW2, placing this value in H0, bringing LW1 into H0, and then performing J65. That is to say, K13 will have the general form:

```
K13      FIND V(LW0,LW2)  
          1LW0  
          J65.
```

In the list of available routines, the compiler finds:

```
J10:  FIND THE VALUE OF ATTRIBUTE (0) OF (1),
```

which may be abbreviated, FIND V((0),(1)). Comparing the arguments of J10 with V(LW0,LW2), we see that (1) must

be set equal to 1W2 and (0) to 1W0. Hence, V(1W0,1W2)
is equivalent to

11W2
11W0
J10.

Hence, the complete code for K13 is:

11W2
11W0
J10
11W1
J65,

and the complete code for J13 is simply:

J13 J52
 11W2
 11W0
 J10
 11W1
 J65
 J32 0 .

THE GENERAL COMPILER

The General Compiler [U135] is an executive routine whose task is to compile a routine from information in any of the forms already discussed (SDSC and DSCN) or in other forms that may be described. It takes as its input the name of the routine to be compiled. Associated with this name (on its description list) is the information to be used in the compilation. More formally:

A routine is a description list containing values of some subset of the following attributes:

1. IPLN--IPL name [X25]. The value of this attribute is a description list that names a region

[X33] and a location [X34] in the region;
e.g., J60, R149, J3.

2. JDEF--IPL-V definition [X22]. The value of this attribute is a list of IPL-V instructions, each in the form of a description list describing [attributes X40-X46] the corresponding IPL-V word, that defines an IPL-V routine with the specified name. For example, the routine with IPLN J3 might have the following JDEF:

```
J3      10J3
        20H5      0 .
```

3. DSCN--Descriptive name [X20]. The value of this attribute is an imperative sentence (encoded as a list structure) that describes [with attribute X30,X31] the process defined by the JDEF. For example, the routine with IPLN J3 has, as already explained, the DSCN:

REPLACE THE SYMBOL IN H5 BY J3.

4. SDSC--State description [X240]. The value of this attribute is a list structure that describes the state of the IPL computer before and after the routine in question has been executed. Only changes are mentioned explicitly. Thus the SDSC of J3 is:

H5: S1,R0. J3,R0.

5. FLWD--Flow diagram [X267]. The value of this attribute is a list structure that gives the flow diagram corresponding to the JDEF. This list structure will be described in more detail later.
6. ASOJ--Associated J definition [X23]. The value of this attribute is the IPL name of a routine associated, in a manner to be described later, with a given routine.

A compiled routine is a routine that has a JDEF. Now we can state the problem of compiling a routine as follows: Given a routine without a JDEF (the present object), find the corresponding routine with a JDEF (the goal object). "Corresponding" means that the compiled routine has the same SDSC or DSCN as the given routine. Figure 1 presents the flow diagram of a compiler [U135] that uses means-end analysis to accomplish this compilation.

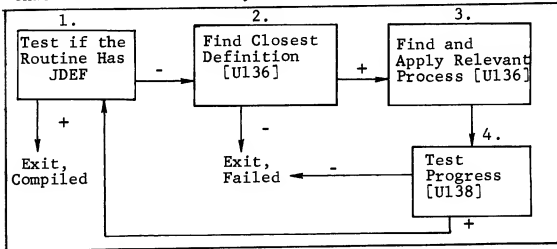


Fig. 1--Flow Diagram for: COMPILE ROUTINE R1

Let us translate this flow diagram into the language of means-end analysis.

1. Test if the routine has JDEF. This test determines whether the present object has the characteristics of the desired object. If so, the compilation is complete.
2. Find the closest definition [U136]. This process corresponds to finding a difference between the present and desired objects. However, we generalize this notion to mean look for a characteristic of the present object that will suggest a relevant operator. If the object possesses a DSCN, then an attempt could be made to compile the JDEF from the DSCN; if it possesses a SDSC, an attempt could be made to compile the JDEF from the SDSC. The attributes that the routine could possess are listed in an order that reflects the relative ease of compiling a JDEF from them. The process then finds the first attribute on this list that the routine to be compiled possesses. In the present form of the compiler, it is assumed that it is easier to compile from a DSCN than from a SDSC; hence, the attributes are listed in this order. If the routine possesses no attribute that can be used as a basis for compilation, the compiler reports a failure.

3. Find and apply the relevant process [U137]. The input to this process is the "closest definition" that has just been found. A table of connections is searched to find a process that is relevant to compiling the JDEF from the closest definition. If one is found, it is applied (in a manner to be described later).
4. Test progress [U138]. If the operator has been applied successfully, the routine will now possess at least one attribute (a JDEF or another) it didn't previously possess. If the progress test detects that it now has a definition closer to the JDEF than any it had previously, it initiates a new compilation cycle; if not, it reports a failure and quits.

The present list of "closest definitions" is very short, consisting only of DSCN and SDSC. The present table of connections [X198] is also brief:

1. If the routine possesses a DSCN, apply the operator, COMPILE JDEF FROM DSCN.
2. If the routine possesses only a SDSC, apply the operator, COMPILE JDEF FROM SDSC.

IV. RELATION OF THE HEURISTIC COMPILER TO GPS

Since each of the major components of the Heuristic Compiler is a system of means-end analysis, each of these components can be viewed as a rudimentary GPS. It should therefore be feasible, by modifying the top-level programs, to bring the Heuristic Compiler into a form that would allow its problem-solving processes to be governed by GPS. The programs for detecting differences, the tables of connections, and the operators would provide definitions of task environments for GPS. To accomplish this, GPS would have to be arranged so that a subproblem could be attached by applying GPS to a new task environment. That is, GPS would first be applied to the task environment of the General Compiler; applying an operator in this environment would consist in applying GPS to the task environment of the DSCN's or the SDSC's, as the case may be.

V. FLOW DIAGRAMS

Up to this point, we have considered only very simple programs, requiring no branches or loops. Each program is a list of instructions; each instruction is an IPL word represented as a description list with the attributes type [X40], name [X41], sign [X42], P [X43], Q [X44], symbol [X45], and link [X46].

To represent a program with branches and loops, we divide the program into segments. Each entry point to a loop (an instruction with a local name) begins a new segment; each branch instruction ($P = 7$) ends a segment. Each segment has the same attributes as an IPL word--specifically: name, P, symbol, link, and an additional attribute [X22], whose value is the list of IPL instructions for the segment. The name of the first instruction of the segment is assigned as the name of the segment; if the segment ends in a branch instruction, it is assigned $P=7$, and its symbol and link are set equal to the symbol and link of the branch instruction. If the segment does not end in a branch, it is assigned $P = 0$ and $SYMB = 0$, and its LINK is set equal to the link of its last instruction. Under these conventions, the list of segments is a flow diagram of the routine with the detail of the routine segments appended.

To illustrate the format of a flow diagram, we show below the code for IPL routine J77, followed by its flow diagram. The DSCN of J77 is: TEST IF THERE IS A SYMBOL EQUAL TO (0) ON LIST (1).

IPL-V CODE FOR J77

J77	J50	90	<u>Segment I:</u> Put (0) in W0.
90	J60		<u>Segment II:</u> Find next
	7091	92	<u>location on list (1).</u>
92	12H0		<u>Segment III:</u> Test if
	11W0		<u>symbol at location is</u>
	J2		<u>equal to (0).</u>
	7090	91	
91	30H0		<u>Segment IV:</u> Clean up
	J30	0	<u>and exit.</u>

FLOW DIAGRAM FOR J77

J77	0	90
90	7 91	92
92	7 90	91
91	0	0

From the description of the flow diagram, it is easy to provide a program [U139] that will construct a flow diagram from an IPL routine, and a program [U133] that will compile an IPL routine from the flow diagram and appended code segments. In this way the task of compiling an IPL routine is reduced to the problem of compiling its flow diagram, and compiling the code for each of the segments of the flow diagram.

The program for compiling such a routine from its DSCN has not yet been written, but examination of the structure of the routine itself shows what is involved. The test involves a quantifier--whether there exists a symbol with a certain property on a particular list of symbols. In IPL-V, such existence tests are performed by means of a loop or a generator; the members of the set in question are produced one by one and tested for their possession of the property. If a test result is positive, the process stops, and the signal, H5, is set plus. If the set is exhausted, the signal, H5, is set minus. Thus, a standard flow diagram can be used for all routines of this kind:

- A Perform required setup.
- B Locate another member of set
(if none, exit, via D),
- C Perform test on member
(if it succeeds, exit, via E;
if it fails, return to B),
- D Exit with signal minus.
- E Exit with signal plus.

Except for the provision of two distinct exits, this is identical with the flow diagram previously shown for J77 (Set A = J77, B = 90, C = 92, D = E = 91). Now, we can compile for each segment of the flow diagram a routine that corresponds to the DSCN of that segment. For example, FIND ANOTHER MEMBER OF (1) becomes J60 (after appropriate recognition of the changed location of (1)); PERFORM TEST ON MEMBER becomes:

12H0
11W0
J2 .

The only complications lie in moving the inputs for the various processes (J60 and J2) in appropriate ways. The compiler can do this in a straightforward, if inefficient, way by using the working storages. Thus, an unedited compiled version of J77 might look like this:

J77	J51	90
90	11W1	
	J60	
	20W1	
	7091	92
92	12W1	
	11W0	
	J2	
	7090	93
91	J31	0
93	J31	0 .

The same flow diagram would be used in the compilation of J62: LOCATE ON (1) AN X SUCH THAT C(X)=(0). In fact, this routine is identical with the one just discussed, except that it requires 11W1 before the exit. It should be observed that the indefinite article "an" plays the same role in the DSCN of J62 as the quantifier "there is a" in J77. The compiler, therefore, would be provided with the knowledge that the above flow diagram, using J60 in the second segment, is the appropriate means for translating this quantifier.

Declarative and interrogative sentences in a DSCN correspond to tests in the compiled routine. Thus, the

phrase, "such that $C(X)=(0)$," leads to the question,
"Does $C(X)$ equal (0) ?" and thence to the test $J_2(C(X),(0))$.

PART II

GENERAL IMPLICATIONS FOR REPRESENTATIONS

Our examination of flow diagrams has already led us to consider how some syntactical devices of English (e.g., the definite and indefinite articles) are to be rendered in the compiler. In this part, we raise at a more general level the question of the syntactical flexibility and range of the compiler languages. In Sec. VI, we ask what forms of English expressions are handled naturally and simply by the DSCN and SDSC languages, respectively. In Sec. VII, we ask how we would go about formalizing the notion of representation so that a problem-solving compiler could be given the task of designing its own representations.

VI. LANGUAGE AND REPRESENTATIONS IN THE COMPILER

We can ask appropriately about any compiler, "What range of source statements can it accept?" In Part I, we have discussed two kinds of source statements, DSCN's and SDSC's. Let us now consider in a little more detail the range of English-language expressions that these compilers can handle. We take up the language of descriptive names and the language of state descriptions in turn.

DESCRIPTIVE NAMES

The DSCN's are particularly interesting because they take the form of English sentences--imperatives, or, as we have just shown, declaratives and interrogatives. How restricted is the language of DSCN's in relation to the whole class of grammatical English sentences?

Consider the sentence, "What is the color of that apple?" The answer might be, "It's color is red," or even, "It is red." In the original sentence, "that apple" denotes a particular object; "the color," an attribute of that object; and "what," the unknown value of the attribute. In the replies, "it" denotes the same object as "that apple"; "red," the (now known) value of the attribute. Thus, we might represent the question and the first answer, respectively, as:

? = Color (that apple)

Red = Color (it).

The second answer can be interpreted as synonymous with the first if we stipulate that "red" can be a value only of the function "color."

The process in IPL-V that provides the answer to such questions is J10: FIND THE VALUE OF ATTRIBUTE (0) OF (1). In terms of our example, this is: "Find the value of attribute 'color' of that apple." Thus, the process that answers the question takes two inputs--the names of the attribute and the object--and produces the value as its output. It defines a function, in the mathematical sense of the term. In the statement of the process, "attribute" is in apposition with "color," the former term specifying the genus to which the argument belongs. We could equally (though not quite grammatically) have said, "of the object, 'that apple.'" Likewise, the phrase, "Find the value of the color," is synonymous with, "Find the color," "value" being in apposition with the (implicit) "?". That is, in English, we abridge, "The value of the attribute, the color, of the object, that apple, is red," to, "The color of that object is red," or, "That object is red." We can do this because "color" is an attribute, "that apple" is an object, and "red" is a value--nothing is added to meaning by making these classifications explicit.

This example shows how, in general, we can handle a wide class of grammatical forms within the framework of the DSCN's. Interrogatives are unknowns, like the x's of algebra, whose genera may be specified, in part or full, to identify them. The couple "the ... of ..." signals a determiner--a phrase that names something by giving enough of its properties to tag it uniquely. Pronouns (e.g., "it") and pronominal adjectives ("that") identify by reference to terms that have occurred in previous sentences. "Find" is a general process that replaces a determiner by the object determined. Appositive phrases and relative clauses provide additional identificatory information about the object to which they refer. Adjectives, adverbs, prepositional phrases (other than "of" phrases), and adjectival nouns have the same function--identification or description. Quantifiers ("there is," "all," "a," "some," etc.) require special treatment--several of them have already been discussed.

The present DSCN Compiler was constructed specifically to handle verbs (processes), determiners (especially those involving "the ... of ..."), and proper names (in IPL-V these are always locations). Essentially, what the compiler does is replace determiners by the appropriate proper names, using the FIND processes for the compilation. In the previous section, we indicated how loops and flow diagrams could be used to handle quantifiers and con-

junctions, including "and," "but," and "if ... then." Pronouns could be handled in a manner similar to that used for determiners. Appositive and modifying words, phrases, and clauses could be used as aids in identifying proper names. It appears that with these extensions, the DSCN format would encompass most of the forms of grammatical English sentences. Only programming, of course, can determine to what extent this claim is correct.

STATE DESCRIPTIONS

Just as the DSCN language admits of considerable flexibility in representing English sentences, so the SDSC language admits of broad flexibility in the representations of information in the computer. This flexibility is achieved by using description lists as the holders of information. Each computer address that is referred to in the SDSC is represented by a cell having a description list. On the description list are the attributes NAME, TYPE, P, Q, SIGN, SYMBOL, LINK--i.e., precisely the attributes that name the fields in an IPL word (and the name of that word) and that appear on the coding sheet. Since the SDSC makes these attributes explicit, the program that uses the SDSC's need not be provided with this information in any format more specialized than the description list format itself. Moreover,

additional attributes and their values can be added, ad lib, to the description list.

By introducing attributes that refer to a particular machine representation of IPL, the SDSC language is readily extended to admit statements about the relation between IPL and its particular representations. For example, suppose the fields in the words of a particular computer were designated by the attributes DECREMENT, ADDRESS, etc. We could then define a machine language representation of IPL by setting up appropriate correspondences between IPL attributes and machine language attributes. (For example, we might specify SYMB = DECREMENT, LINK = ADDRESS, etc.) We shall indicate in the next section how this technique can be used to give the Heuristic Compiler the capability of designing appropriate representations--hence, how the compiler itself might choose an appropriate machine language representation of IPL prior to undertaking the task of compiling IPL into a machine language program.

There is also no necessity in SDSC that each description list should give the description of the contents of a single computer address. Alternatively, it may describe a whole list. Suppose, for example, we wish to represent the fact that the symbol in cell H5 is J3, and that H5 is linked to a pushdown list having unknown con-

tents. All of this information can be given in two description lists:

- | | | | | |
|----|------------|------------|------------|---------------------------------|
| 1. | NAME
H5 | SYMB
J3 | LINK
R0 | KIND OF OBJECT
Pushdown cell |
| 2. | NAME
R0 | | | KIND OF OBJECT
List |

We have already illustrated how this flexibility is used to represent segments of instructions in flow diagrams.

VII. THE DESIGN OF REPRESENTATIONS

One distinction between the restricted, relatively simple tasks we call "coding," and the broader, more difficult tasks we call "programming," is that the latter may encompass the selection or design of an appropriate problem representation, while the former do not. Our discussion of languages now enables us to see what is involved in the design or selection of a representation, and what we would need to do in order to give the Heuristic Compiler the capacity to grapple with such design and selection tasks. To illustrate this point, we shall take an example of a representation problem within the structure of IPL-V itself.

Let us suppose that we had an operating "basic" IPL system, quite like the language defined in the IPL-V Manual, except that the description list processes were omitted. We now give a programmer the task of introducing description lists into the language, using the basic system itself to define them, without writing any new machine code.

What do we mean by "introducing description lists"? We mean that we wish to be able to associate with the name of an object (which in IPL is always an address) a description of that object. The description consists of a set of pairs: one member of each pair is an attribute;

the other member of the pair is the value of that attribute for the object to which the description belongs. Moreover, we wish to be able to store and retrieve descriptive information about objects. That is, we wish to be able to add new pairs to descriptions, and when we are given the name of an object and an attribute, we wish to be able to find the value of that attribute for that object. Stated formally:

With every object A_i , we associate a set of pairs, (B_j, C_{ij}) . The number of pairs is to be arbitrary and variable, and we want a process that will answer questions of the form: $? = B_j(A_i)$.

How could a programmer solve this problem? By the basic conventions of IPL-V, "object" already means "address." Thus, he must find some way of associating a set with each address. Again, in IPL, the standard way to represent a set is by a list. The question then becomes, "What list can we associate with an address?" The basic relations that are represented in IPL-V are CONTENT OF and NEXT. The CONTENT OF a cell is the SYMB of that cell, and the NEXT of the cell is its LINK. Moreover, basic processes exist for FIND CONTENTS ($P = 1, Q = 1$) and FIND NEXT (J60). Thus, the set we associate with an address can be taken to be the list whose name is the SYMB (or, alternatively, the LINK) at that address. Let

us call this list (pursuing the first alternative) the description list associated with the address.

We must next define a format for description lists that will represent the pairing of attributes and values. One method would be to associate a pair of words with each element of the description list, again using the relations CONTENT and NEXT. Thus, if S_j is the content of the j^{th} number of the description list, we could define $B_j = \text{CONTENT OF } S_j$, and $C_{ij} = \text{CONTENT OF NEXT OF } S_j$. (This is substantially the representation that was used in an earlier version of IPL. An even simpler representation would make $C_{ij} = \text{LINK OF } S_j$.) Now, to add a pair, B_j, C_{ij} , to the description list of A_i , we add a cell to the description list, assign it a SYMB (S_j), assign B_j as the SYMB of S_j , and assign C_{ij} as the SYMB of the LINK of S_j . Similarly, if we are given A_i and B_j , to find C_{ij} , we first find the description list, CONTENT OF A_i , and we go down this list comparing B_j with the CONTENT of the CONTENT of each location on the list. When we obtain a match, we find the CONTENT of the cell next to the matched cell, and this is the desired value. These processes follow from the representation.

An alternative representation is obtained by dividing the members of the description list into two subsets--its ODDS and its EVENS. We then take the ODDS as the attributes; the value of an attribute is simply the EVEN that follows

it. These definitions, again, make use only of the relations CONTENTS and NEXT. Thus, an ODD of a list is the FIRST of the list or the NEXT of the NEXT of an ODD.

This definition allows us to construct a loop that will find, in sequence, all the attributes on the list. Given the location of an attribute, a FIND CONTENTS OF NEXT finds its value. This representation is, of course, the one actually adopted in IPL-V.

It will be instructive to see what the program for J10; FIND THE VALUE OF ATTRIBUTE (0) OF (1), looks like in each of these representations. We write the two programs side by side:

J10	J51	J10	J51	
	12W1		12W1	Find description list.
92	J60	92	J60	
	7090		7090	
	60W1		60W1	
	52H0		52H0	Find next attribute.
	12H0		11W0	
	11W0		J2	Test if it is equal to (0).
	J2		7091	
	7091		11W1	
	J60		J60	If so, find value.
	52H0 J31		52H0 J31	
91	30H0	91	11W1	<u>list.</u>
	11W1	92	J60	Proceed down description/
90	30H0 J31	90	30H0 J31	Exit, attribute not found.

We see from this example that designing a suitable representation amounts to finding an isomorphism. A "description" was defined in terms of certain elements (objects, attributes, values), relations between elements

(e.g., the attributes of an object), and processes (e.g., FIND THE VALUE OF (0) OF (1)). The programmer had to find a set of elements, relations, and processes defined in IPL-V that were isomorphic with the required elements, relations, and processes. I have not worked out how such a search could be automated, but the main requirements are clear. In particular, to enable the Heuristic Compiler to perform this search, it would have to be provided with lists of the available elements, relations, and processes, or it would have to be able to recognize such things when they were described in the DSCN or SDSC languages. For example, it would have to recognize that every determiner (e.g., "the ... of ...") defines a relation.

As a second example of what is involved in designing a representation, let us consider the representation of IPL-V words in SDSC. Since each word consists of a number of symbols belonging to different fields, we can again use the description list format, in which we equate "field" with "attribute" and "symbol in field" with "value of attribute." But the symbols in the NAME, SYMB, and LINK fields themselves contain encoded information, for they are in the form ANNNN, where A is an alphameric symbol, and NNNN is a number. Hence, we represent each of these symbols by an object with a description list containing the attributes REGION and LOCATION. The value of REGION is the alphameric symbol A; the value of LOCATION, the

number NNNN. Therefore, the information that a "local symbol" is one with $A = 9$ can be represented. Again, the lesson here is that we must create an isomorphism between the elements of the representation, their relations, and the structure to be represented.

Suppose, as a third example, that we set ourselves the task of writing a program to sort a bridge hand. To accomplish this task the meanings of "sort" and "bridge hand" must be known. A bridge hand is a set of (13) elements, each characterized by a primary characteristic, suit (4 possible values), and a secondary characteristic, denomination (13 ordered values). Sorting means ordering a set of elements by one or more characteristics, taking account of the ordering of values where this is defined.

In this case we find a straightforward isomorphism: Each element in the bridge hand is to be represented by an object having a description list with attributes SUIT and DENOMINATION. A sorted bridge hand is to be represented by a list of such elements, with the ordering of the list to correspond with the ordering of the sort. It now becomes a straightforward (if difficult) compiling job to write a SORT routine that will produce a list with these properties. Moreover, if it is done correctly, it should be possible to write the routine in the generalized form: SORT (0) IN FORMAT (1), where (1) enumerates the

attributes, their ordering, and the orderings of the values that define the sorted object.

We shall explore this particular scheme in more detail in Sec. VIII of Part III. Perhaps enough has been said here to demonstrate that selecting or designing a representation is a problem-solving task that can be attacked with the same general kinds of heuristic techniques as other problem-solving tasks.

PART III

EXPERIMENTS WITH REPRESENTATIONS

In Part III, we propose some extensions of the Heuristic Compiler, most of which are motivated by linguistic considerations. In particular, we explore some methods for enabling the compiler to handle input statements in forms that are close to natural language.

There are a number of important respects in which natural languages differ from the usual programming languages. We shall be especially concerned with three of these differences:

1. In natural languages, the word is the most important unit of meaning. (For the moment, we do not need to distinguish among "word," "morpheme," and "idiom.") In most computer languages, the sentence (usually an imperative sentence, called an "instruction") is the basic unit of meaning. Thus, if a person understands, separately, the verb "sort," and the noun phrase "bridge hand," he can probably obey the instruction, "Sort the bridge hand." In most computer languages, a compiler would not be able to assemble, "Sort a bridge hand," from "sort" and "bridge hand," but would have to be provided with a number of specialized sort routines.

2. In natural languages, most communication makes use of sentences in the indicative or declarative mode.

In computer languages, most sentences are in the imperative mode. Computer languages are primarily languages of command, and not languages of information, description, or advice.

3. In natural languages, many alternative sentences can be phrased that "mean" about the same thing. The recipient of a natural-language communication is able to decode the communication without too much concern for details of format. In computer languages, there are various harassing constraints on format. Failure to observe these constraints usually causes an error condition.

These differences are, of course, differences of degree and not of kind. Moreover, research on computer languages over the past decade has already made substantial progress toward decreasing or erasing them. Basic processes have become more general and parameterized; various forms of declarative statements have been introduced; compilers have been designed to accept relatively informal input statements. The gap, however, between natural language and computer languages is still large, and annoying to those who are engaged in man-machine communication.

We are here concerned with extensions of the Heuristic Compiler directed towards reducing these

differences.* Section VIII indicates how generalized processes (verbs) can be introduced into the DSCN Compiler. Section IX describes subroutines for storing and retrieving descriptive information in declarative sentences. Section X describes an approach toward natural-language flexibility in input statements for the SDSC Compiler, making use of the descriptive information provided by the techniques of Sec. IX.

*The thinking reported here, particularly in Sections IX and X, was greatly stimulated by the work of Robert K. Lindsay, "The Reading Machine Problem" (Unpublished Ph.D. Thesis, Carnegie Institute of Technology, 1960), which has been revised as Toward the Development of a Machine Which Comprehends, University of Texas, Austin, May 1961. I owe a great deal also to stimulating discussions with Hugh Kelly and Allen Newell of The RAND Corporation and Carnegie Tech, respectively.

VIII. GENERALIZED PROCESSES

We shall consider two classes of processes: one designated by the verb "find," the other by the verb "sort."

"FIND" PROCESSES

The possible interpretations of FIND are numerous to the point of being meaningless. Any routine that takes some symbolic structures as inputs and produces one or more other structures as outputs may be called a process for "finding" the latter. Thus: FIND SINE A, FIND THE STATE DESCRIPTION OF ROUTINE K, FIND THE PROOF OF THEOREM T. From this point of view, a program like the General Problem-Solver is simply a fairly general FIND routine.

Hence, any routine flexible enough to interpret correctly the verb "find" wherever it occurs in normal English prose, would have to make considerable use of context. In the present section we shall aim at a lesser degree of flexibility. Consider the two classes of processes typified by:

FIND THE state description OF routine K, and
FIND A ON

The first example designates an object associated in a particular way with a specific object, K. In IPL-V there is provided a special format, the description list, for holding such information in memory, and a set of

processes, J10 to J16, for entering and retrieving the information. Section IX examines definite descriptions of this special kind at length. In the present section, then, we shall limit ourselves to the verb "find" as it occurs in instructions like, FIND A ... ON ..., and FIND THE ... ON Even this scheme covers a considerable variety of processes:

FIND A symbol, S10, ON list L.

FIND AN object whose type is A4 ON list structure L.

FIND THE third symbol ON list L.

FIND THE largest integer data term ON the lists of list L.

In the first two examples, the indefinite article indicates that the object sought is not necessarily unique; in the last two examples it is. In the first two examples the properties that define the object sought are absolute--their presence or absence can be ascertained without reference to other objects. In the last two examples, the properties are relative; and indeed, in the fourth example, the entire set of objects must be examined before the one sought can be identified. In the first example, the object sought is designated by proper name (and the information is added that it is a "symbol"). This example can be approximated to the others by rephrasing it: FIND A token equal to S10 ON list L.

With these preliminaries out of the way, we shall describe in detail a rather general FIND process. The object sought will be specified by some sublist of the symbols on the pushdown list of the communication cell, H0. The specification of the place to be searched will be given by other symbols on that pushdown list. Hence, we can symbolize the desired process:

FIND A $F_1[(0), (1), \dots, (k)]$ IN $F_2[(k+1), (k+2), \dots, (n)]$,
 where (0), (1), etc., designate, as usual, the symbols on the pushdown list of H0.

We suppose that the compiler is provided with a lexicon that contains, among others, the words "FIND," " F_1 []," and " F_2 []." The lexical entry for each of these is a description list containing, for "FIND" and " F_1 ," the attribute IPL ROUTINE, and for " F_2 ," the attribute TYPE OF OBJECT.

The IPL ROUTINE associated with FIND in the lexicon will contain certain variables, to be replaced by constants derived from an examination of F_1 and F_2 . We will first present the routine, as it would appear in the lexicon, and then explain the meaning of certain of the symbols in it.

FIND	J5n	
	11W(n-k)	
	1090	
	GEN (F_1, F_2)	
	J5	J3n
90	40H0	
	11Wk	
	TEST(F_1)	
	J5	
	7090	J8

As previously indicated, k is the number of arguments in F_1 , (n-k) the number of arguments in F_2 , and n the number of arguments in the two functions taken together. Thus J5n preserves W_0 to W_n and moves all the arguments of F_1 and F_2 into the W 's; J3n restores W_0 to W_n . The instruction $11W(n-k)$ brings into H_0 the arguments of F_2 , while $11Wk$ brings in the arguments of F_1 . The numbers k and n-k are to be determined, of course, by examining F_1 and F_2 . The variable, $GEN(F_1, F_2)$, is to be replaced with a generator obtained from the lexical entry for F_2 , while $TEST(F_1)$ is to be replaced with a test associated with the lexical entry for F_1 . All the other symbols have their usual IPL-V meanings.

We can now compile FIND A F_1 IN F_2 as follows:

1. Get IPL ROUTINE of FIND.
2. Supply values for k, n, and n-k where required.
3. Make list of subroutines required ($GEN(F_1, F_2)$, $TEST(F_1)$).
4. Get the IPL ROUTINE of TEST, supply it with its arguments, and insert it in the IPL-V code for FIND.
5. Get TYPE OF OBJECT of F_2 , and get the associated GEN for that type in the dictionary of generators; insert it in the IPL-V code for FIND.

A word may be added to this account to indicate how relative properties are handled in FIND THE THIRD ..., or FIND THE LARGEST. In these cases, the test needs to be based on a recursive process--in the case of FIND THE THIRD ..., a counting process; in the case of FIND THE

LARGEST ..., a process that resets LARGEST SO FAR equal to the larger of LARGEST SO FAR and the current integer. The test--the function F_1 --would, in these cases, make provision for storing in the W's the intermediate products of calculation.

"SORT" PROCESSES

As our second essay toward generalized processes, we take the verb "sort." Suppose that the objects we wish to sort are description lists. It is easy to construct a general routine, SORT (0) ON (1), where (0) is the list of objects to be sorted and (1) is a specification of the attributes, their ordering, and the orderings of attribute values on which the sorting is to be based. Thus, if (0) were a list of description lists representing the cards in a bridge hand, (1) would specify that the attributes are suit and denomination (in that order), that the suit values are S,H,D,C, in that order, and the denomination values A,K,Q,J,10,9, etc., in that order. A further step toward generalization would allow the sort routine to be compiled from a definition of the collection of objects to be sorted. Let us see how this can be done when the object to be sorted is a bridge hand. First, we store in memory description lists providing information about the terms LIST, DESCRIPTION LIST, and ATTRIBUTE:

LIST
TYPE: Class of objects
ATTRIBUTES: Type of members

DESCRIPTION LIST
TYPE: Class of objects
ATTRIBUTES: Attributes

ATTRIBUTE
TYPE: Class of symbols
ATTRIBUTES: Values.

That is to say, a list is a class of objects; the description list for any class of objects of type LIST will have the attribute CLASS OF MEMBERS. A description list is also a class of objects; the description list for any class of objects of type DESCRIPTION LIST will have the attribute ATTRIBUTES. An attribute is a class of symbols; the description list for any class of objects of type ATTRIBUTE will have the attributive VALUES.

Next, we store in memory description lists providing information about the terms BRIDGE HAND, CARD, SUIT, and DENOMINATION.

BRIDGE HAND
TYPE: List
MEMBERS: Cards

CARD
TYPE: Description List
ATTRIBUTES: Suit, denomination

SUIT
TYPE: Attribute
VALUES: Spade, hearts, diamonds, clubs

DENOMINATION
TYPE: Attribute
VALUES: A,K,Q,J,10,9,8,7,6,5,4,3,2

Now we can compile SORT THE BRIDGE HAND (0), as follows: From the information just stored, we find that BRIDGE HAND has the type LIST. From LIST, we find that BRIDGE HAND will have the attribute TYPE OF MEMBERS. Finding the value of this attribute, we determine that the members of bridge hands are cards. CARD is a description list, which has the attribute ATTRIBUTES. The attributes of cards are suit and denomination. We would therefore compile the sort routine to sort on suit and denomination, in that order of priority. Examining the values of these two terms, in turn, we find the order in which these values are to be arranged in sorting.

If it were known to the program that a bridge hand is a list of description lists, then the sorting routine could obtain the information about attributes and values by direct examination of one or more examples of a sorted bridge hand, and without being given the information about CARD, SUIT, or DENOMINATION explicitly. If the examples were not too special (e.g., a hand of thirteen spades), the program could determine what attributes a card possessed, which of these was relevant to the sorting, the hierarchy of attributes, and the ordering, if any, of the values of each. This information could then be used to compile the specific sorting routine required.

Thus, we see that the key to providing a generalized routine for a verb like SORT lies in providing syntactically or semantically the information needed to supply the routine with the parameter values it requires. This can be accomplished (syntactically) through a scheme of declarative sentences that describe the objects under consideration; or (semantically) by providing examples that can be analyzed. Moreover, the description itself can be generated inductively from examples.

The description that would allow compilation of the "sort" routine could also be used to compile TEST IF X IS A BRIDGE HAND. Thus, storing descriptions of classes of objects is an important means for factoring sentence meanings of nouns and verbs, respectively. It provides a powerful basis for introducing general processes.

RECURSIVE FUNCTIONAL LANGUAGES

Consider an instruction language consisting of a set of functions each admitting as arguments the values of functions of the set. Each of the functions can be regarded as a FIND instruction--i.e., FIND THE VALUE OF F FOR THE GIVEN ARGUMENT VALUES. It may be executed recursively by finding, first, the values of each of its arguments, then using these to compute the value of the function. Hence, the interpreter of such a language may itself be regarded as a generalized FIND instruction.

Next, consider an instruction like SORT OBJECT A, in a recursive functional language. The definition of SORT, if it is a generalized routine like those described in the previous section, may read something like ARRANGE ACCORDING TO THE ATTRIBUTE VALUES OF. The definition has different arguments from SORT itself--it refers to the attribute values of A instead of the object A. The interpreter would need to be general enough to replace SORT A by ARRANGE A BY THE ATTRIBUTE VALUES OF A; then execute FIND THE ATTRIBUTE VALUES OF A, and insert the value of this function as the second argument of the ARRANGE function. We have already indicated how the FIND might be accomplished.

With a little further generalization, the scheme could handle apposition--e.g., SORT THE BRIDGE HAND, A. The phrase in apposition would provide information about the type of the object designated, and as we have seen, this information could be used to find the other argument of the ARRANGE routine. Further light will be cast on apposition in the next section, where we shall discuss modifiers that identify an argument, and their relation to modifiers that describe the object.

IX. DEFINITE DESCRIPTIONS

The meaning of much descriptive and expository prose can be captured in a fairly simple language--a sub-language of English--that uses only the verb "is" and noun phrases with definite or indefinite articles in subjects and predicates. Consider the following example, which, while it does not fit this restricted form exactly, is not far from it:

The state description of a routine consists of a list of affected cells. For each affected cell on the list, the state description specifies its input state and its output state.

SYNTACTIC CHARACTERIZATION

We might proceed to formalize this description in either of several ways. I shall call the first of these syntactical, since it makes statements about the terms "state description," "affected cells," "input state," and "output state." These statements can then be stored in association with the relevant terms in a lexicon.

The type of "state description" is "description list";
The attribute of "state description" is "list of affected cells";
The type of "list of affected cells" is "list";
The type of "affected cell" is "description list";
The attributes of "affected cell" are "input state" and "output state."

We have already seen this kind of description in our discussion of generalized SORT routines in the previous

section. Readers who are familiar with the notation known as Backus normal form will observe that a syntactical description of this kind could without much difficulty be translated into that form--or a slight extension thereof. Our interest, however, is in staying close to natural English.

SEMANTIC CHARACTERIZATION

An alternative, semantic, formalization characterizes a given type of object (STATE DESCRIPTION in this instance) by describing an example:

```
X1 is a state description of a routine if
      there are an X0, X2, X3, X4,
      and X5, such that:
X0 is a routine;
X1 is the state description of X0;
X2 is the list of affected cells of X1;
X3 is a member of X2;
X4 is the input state of X3;
X5 is the output state of X3.
```

Properly interpreted, the example implies the syntactic description we gave previously. Consider, for instance, "X3 is a member of X2." With the convention that only lists have members, this statement implies that X2 is a list. From the previous statement, "X2 is the list of affected cells of X1," we observe that this list is the value of the attribute, LIST OF AFFECTED CELLS, of X1.

We can store the example in memory by storing a description list, X0, with attribute STATE DESCRIPTION

having value X1. X1, in turn, is a description list, with attribute LIST OF AFFECTED CELLS having the value X2. X2 is a list whose sole member is X3. X3 is a description list with attribute INPUT STATE having value X4, and attribute OUTPUT STATE having value X5.

ANNEXING DESCRIPTIVE INFORMATION TO AN INFORMATION STORE

Let us use symbols from the X region--e.g., X114, X33--to designate nouns. These nouns will be either proper names (of objects to be represented by lists or list structures) or attributes. Consider now the sentence:

"X114 is the X33 of the X25 of X105."

In this sentence, which is grammatical if inelegant English, "X114" and "X105" are proper names, while "X33" and "X25" name attributes. The objects referred to in this sentence are X114 (or synonymously, the X33 of the X25 of X105), the X25 or X105, and X105. The problem of annexing the information provided by this sentence to an existing memory store depends on what is already in the store.

Suppose, as a first possibility, that no information has been stored previously about the objects mentioned in this sentence. We store the new information by creating a name, call it X200, and assigning it as the value of attribute X25 of X105. Then we assign X114 as the value of attribute X33 of the newly named object, X200.

Suppose, however, that we had previously stored in memory the information that an object named X130 was the value of attribute X25 of X105. Then, to store the new information, we would first have to find X130, and then assign X114 as the value of attribute X33 of X130.

In the first case, we annexed the new information by two ASSIGN processes--in IPL-V, two applications of J11. In the second case, we annexed the new information by a FIND process (J10 in IPL-V) followed by an ASSIGN. We can write a general routine to accomplish this. In processing a sentence like the one we are using as example, we start at the extreme right and search in memory for the object named. If we find it, we proceed to the left, find the first attribute, and find the value of this attribute of the object. If the value exists, it becomes a new object on which we can repeat the process, moving to the next attribute to the left.

When we fail to find an object meeting the description (when J10 fails), we enter a second phase. We now proceed from the left-hand side of the sentence, creating names for new objects as these are needed, and annexing their descriptions to them (by J11), until we reach an object that is already mentioned in memory.

Thus, depending on what is already stored in memory, the same piece of information in the input sentence can

serve either as a descriptive phrase, providing new information to be annexed to the memory structure, or as an identifying phrase, to be used in locating the place in memory where the new information is to be annexed.

In this scheme ambiguity is entirely possible. It can enter because the scheme allows indefinite, as well as definite description. Again, an example will make the point clear. We consider the following sequence of four input sentences:

X114 is the X33 of the X25 of X105.
X115 and X116 are the X99 of the X34 of the
X25 of X105.
X125 is the X41 of a member of the X71 of the
X24 of X105.
X117 is the X75 of the member whose X41 is
X125 of the X71 of the X24 of X105.

Suppose we begin with no information about X105 in memory. Then, as we have seen, the first sentence is stored by two executions of J11. The first creates a new object, say X200, and assigns X114 as its X33; the second assigns X200 as the X25 of X105.

The second sentence is stored by two executions of J11 and one of J10. Working from the right, by J10, X200 is found to be already in memory as the X25 of X105. But there is no value for attribute X34 of X200. Hence, a list, say X201, is created whose members are X115 and X116, and X201 is assigned as the value of X99 of another new object, say X202. Finally, X202 is assigned as the X34 of X200.

Storing the third sentence brings about the creation of X203, whose X41 is X125; of a list X204, of which X203 is member; and of an object, X205, whose X71 is X204, and which is, in turn, the X24 of X105.

The fourth sentence introduces a new complication. It refers to "the member of the X71 of the X24 of X105 whose X41 is X125." A series of J10's will find the X71 of the X24 of X105--that is, X204. The sentence now calls for locating that member of X204 whose X41 is X125. From the previous paragraph, we see that the object in question is none other than X203. Thus X117 would be assigned as the X75 of X203.

Now let us return to the question of ambiguity. Suppose that the fourth sentence read:

X117 is the X75 of the member whose X76 is
X130 of the X71 of the X24 of X105.

Now, when we examine X204, we find, as before, that it has a member, X203. But we have no information to tell us whether or not X203 is "the member of X204 whose X76 is X130." Hence, we do not know whether to assign X117 as the X75 of X203 or to create a new member of the list X204--say X205, and assign X117 as the X75 of X205.

The ambiguity is not a consequence of the particular annexing scheme we have used, but resides more deeply in the nature of things. Let r and s be relations; A , B , C , and X , objects. Suppose we know that $A \mathbf{r} B$ and that there

exists a Y such that CsY and YrB. Then we can neither affirm nor deny that Y is identical with A. (The same difficulty arises in R. Lindsay's program for annexing genealogical information to a family tree.* If we know only that Isaac is a son of Abraham, and Jacob is a son of a son of Abraham, then we do not know whether or not Jacob is a son of Isaac.) Our only recourse is to arrange the annexing routine so that, when it detects such an ambiguity, it outputs an appropriate question.

A program for an annexing routine is given in detail in Appendix B. It will not deal with the ambiguity problem just discussed, but is in all other respects capable of annexing to memory the contents of sets of sentences of the kinds we have been considering.

*Op. cit., p. 43.

X. COMPILATION OF ROUTINES FROM PARTIAL DESCRIPTIONS

One part of the flexibility of natural language depends on the problem-solving capacity of the listener. Information already in his memory allows him to supply details that are omitted from, or only implicit in, the communication. Suppose we wish to give a (human) programmer the task of coding a routine in IPL-V for J3. We might tell him:

"Write a routine, J3, that changes the contents of cell H5 from [S1,R0] to [J3,R0]."

The programmer, familiar with IPL-V, knows that H5 is a so-called "pushdown cell," and that what is wanted is to replace the contents of the SYMB field of that cell, whatever they may be, by the symbol "J3."

What instruction would we have to give to the SDSC Compiler to induce it to perform the same task? The instructions would read:

"Compile the IPL-V definition of the following routine: The NAME of the routine is J3. The LIST OF AFFECTED CELLS of its STATE DESCRIPTION has a member, whose NAME is H5, whose INPUT STATE is the list, S1, R0, and whose OUTPUT STATE is the list, J3, R0."

Given some such statement, the annexing routine described in the previous section could construct an appropriate description list as input to the SDSC Compiler. In Part I, we showed how the SDSC Compiler could then write the desired routine from its state description.

Now the instructions to the compiler have required about three times as many words as the instructions to the human programmer. The reason is easy to see. The essential information to be provided is that the cell to be changed is H5, and that the change is to replace an unknown symbol by the symbol J3. The compiler cannot receive this information without explicit mention that J3 is the first symbol in the list of the output state of affected cell H5 of the list of affected cells of the state description of the routine named J3. The human programmer is capable of supplying this additional information, because he knows what the structure of a state description is like.

The last observation suggests that we might give the compiler the same capability by providing it prior information about the structure of a state description, and allowing it to fill in the implicit detail. Let us employ the language of Sec. IX to see how this might be done.

The complete description list, necessary to the Heuristic Compiler, is equivalent to the following set of sentences:

- Let X100 be a routine whose NAME is J3.
- Let X101 be the STATE DESCRIPTION OF X100.
- Let X102 be a member, whose NAME is H5, of
the LIST OF AFFECTED CELLS OF X101.
- Let S1, R0 be the INPUT STATE OF X102.
- Let J3, R0 be the OUTPUT STATE OF X102.

In parallel fashion, we can store in memory a "template" for the description of a routine. We will use symbols from region "Y" to denote variables.

Y1 is the NAME of Y2.
Y3 is the STATE OF DESCRIPTION of Y2.
Y4 is the LIST OF AFFECTED CELLS OF Y3.
Y5 is a typical member of Y4.
Y6 is the NAME of Y5.
Y7 is the INPUT STATE OF Y5.
Y8 is the OUTPUT STATE OF Y5.
X7 and Y8 are lists.

Now to fit the specific example to the template, we identify J3 with Y1, X100 with Y2, X101 with Y3, X102 with Y5, and so on. Suppose, however, the example were incomplete, as follows:

X100 is a routine whose NAME is J3.
N101 is a cell whose NAME is H5.
S1, R0 is the M1 of X101.
J3, R0 is the M2 of X101.

Here "M1" and "M2" designate attributes whose meanings are not given in the lexicon. It should not be too difficult to devise a process for matching this description with the template. The matching process would discover that X101 has to be identified with Y5, and X100 with Y1. New objects could then be created to correspond with Y3 and Y4, and the appropriate description list stored in memory. The SDSC Compiler, taking this description list as its input, could now compile the code for J3.

Our basic proposal, then, for compiling routines from partial descriptions rests on two devices: (1) the

use of an annexing routine like that described in Sec. IX as a means for constructing description lists from expository sentences; (2) the use of templates to provide information that is not given explicitly in the input sentences. The routine for accomplishing the second task has not yet been written.

Appendix A

PROGRAM LISTING OF THE HEURISTIC COMPILER

This listing includes a rather simple form of the General Compiler [U135], DSCN and SDSC Compilers [U134 and U140, respectively], routines for assembling flow diagrams [U133], and routines for printing JDEF [U125], DSCN [U126], SDSC [U127], and all three of the above [U128]. These routines are called in by a general executive [T1].

List structures in regions E, T, and U, and X5 and X7 are routines. The remaining list structures in region X are data. The version of T1 that is listed will print X102, then compile and print X105 and X100, compile X182 from its flow chart, printing the result, and reconstruct the flow chart of X102 from its code.

The program is in IPL-V (Information Processing Language-V) and is machine independent.

2 A0		0200	
2 E0		0200	
2 L0		0200	
2 N0		0200	
2 T0		0200	
2 U0		0200	
2 X0		0200	
5			
E4.	PRINT LIST (0) OF DATA TERMS WITHOUT NAMES. ENTER LIST NAME		
E4	40H0		E4 000
	J156		E4 010
	E54	91	E4 020
91	J60		E4 030
	7092		E4 040
	12H0		E4 050
	J157	91	E4 060
92	E53	J8	E4 070
E8	1090		E8 000
	J101	E50	E8 010
90	70	91	E8 020
	12H0		E8 030
	J156		E8 040
	7092		E8 050
	30H0		E8 060
	E54		E8 070
	7093	0	E8 080
91	E50		E8 090
	J156	E53	E8 100
92	E50		E8 110
	10N10		E8 120
	J161		E8 130
	52H0		E8 140
	J156	E54	E8 150
93	E50		E8 160
	10N10		E8 170
	J161	J4	E8 180
E40.	MARK ROUTINES ON (0) TO TRACE		
E40	1090		E40 000
	J100	0	E40 010
90	J147	J4	E40 020
E41.	MARK ROUTINES ON (0) NOT TO TRACE		
E41	1090		E41 000
	J100	0	E41 010
90	J149	J4	E41 020
PRINT, CLEAR	E50	J155	E50 000
SPACE BAR ONCE	E51	10N1	E51 000
SPACE BAR TWICE	E52	10N2	E52 000
PERIOD, SPACE TWICE	E53	1091	E53 000
		J157	E52
	91	+21.	E53 010
	E54	1091	E53 020
COMMA, SPACE ONCE		J157	E54 000
	91	+21.	E54 010
	E55	E50	E54 020
E55.	PRINT LIST STRUCTURE, SAVING FOR NEXT PROCESS		
	40H0	E8	E55 000
T1	3J0		E55 010
	10A99		T1 000
	J154		T1 010
			T1 020

	13A0			
	10930			
	J100			
	10X102	T1		
	U128	T1		
	10X105	T1	030	
	40H0	T1	040	
	U135	T1	050	
	30H0	T1	060	
	U128	T1	070	
	10X100	T1	080	
	40H0	T1	090	
	U135	T1	100	
	30H0	T1	110	
	U128	T1	120	
	10X182	T1	130	
	10X41	T1	140	
	J10	T1	150	
	10X180	T1	160	
	10X181	T1	170	
	U133	T1	180	
	J50	T1	190	
	J90	T1	200	
	40H0	T1	210	
	11W0	T1	220	
	10X22	T1	230	
	J11	T1	240	
	40H0	T1	250	
	U128	T1	260	
	40H0	T1	270	
	U139	T1	280	
	40H0	T1	290	
	10X26	T1	300	
	J10	T1	310	
	40H0	T1	320	
	U125	0	T1	330
930	J147	0		
T99	J166	J165	T99	000
CONSTRUCT IPLV WORD (3), WITH	U100	J52	U100	000
		J90	U100	005
		J136	U100	006
P=(0),Q=(1),SYMB=(2). NO OUTPUT		40H0	U100	010
		11W0	U100	020
ASSIGN P		10X43	U100	030
		J11	U100	040
		40H0	U100	050
		11W1	U100	060
ASSIGN Q		10X44	U100	070
		J11	U100	080
		40H0	U100	085
		11W2	U100	090
ASSIGN SYMB		10X45	U100	100
		J11	U100	110
CONSTRUCT IPLV SYMBOL, WITH	U101	J51	U101	000

REGION=(0),LOCATION=(1). OUTPUT	J90		U101 005
(0) IS LOCAL NAME OF SYMBOL	J136		U101 010
	40H0		U101 020
	11W0		U101 030
ASSIGN REGION	10X33		U101 040
	J11		U101 050
SAVE LOCAL NAME	40H0		U101 055
ASSIGN LOCATION	11W1		U101 060
	10X34		U101 070
	J11	J31	U101 080
CONSTRUCT DESIGNATED SYMBOL,	U102 J51		U102 000
(0)=Q, (1)=SYMB. OUTPUT (0) IS	J90		U102 010
LOCAL NAME OF D.SYMB	J136		U102 020
	40H0		U102 030
	11W0		U102 040
ASSIGN Q VALUE	10X44		U102 050
	J11		U102 060
	40H0		U102 070
	11W1		U102 080
ASSIGN SYMB VALUE	10X45		U102 090
	J11	J31	U102 100
CONSTRUCT - INPUT D.S.,(0) IS D.S.	U103 10N1		U103 000
NO OUTPUT	10X43	J11	U103 010
CONSTRUCT - PROCESS,P=(0),	U104 J50		U104 000
DES.SYMB.=(1). OUTPUT (0) IS LOCAL	40H0		U104 010
NAME OF PROCESS	11W0		U104 020
	10X43		U104 030
	J11	J30	U104 040
CONSTR. J-DEF.,NAME (0),OF (1).	U105 10X41	J11	U105 000
CONSTRUCT PROCESS, P=0, Q=0,	U106 J50		U106 000
(0)=SYMB. OUTPUT (0) IS LOCAL	J90		U106 010
NAME OF PROCESS	J136		U106 020
CREATE WORD	40H0		U106 030
AND MAKE (0) ITS SYMB	11W0		U106 040
	10X45		U106 050
	J11	J30	U106 060
COMPARE IPL SYMBOLS (0) AND (1)	U107 J51		U107 000
FOR IDENTITY. SET H5	11W1		U107 010
	10X33		U107 020
	J10		U107 030
	11W0		U107 040
	10X33		U107 050
	J10		U107 060
TEST IF REGIONS EQUAL.	J114		U107 070
	70J31		U107 080
IF SO,	11W1		U107 090
	10X34		U107 100
	J10		U107 110
	11W0		U107 120
	10X34		U107 130
	J10		U107 140
TEST IF LOCATIONS EQUAL.	J114	J31	U107 150
REPLACE ARGS. IN ROUTINE (0)	U108 J40		U108 000
WITH COPIES.	U114		U108 010

FIND NEXT ARG.	90	J60		U108 020
IF NONE, EXIT		7091		U108 030
		60W0		U108 040
COPY ARGUMENT, AND		52H0		U108 050
		J74		U108 060
INSERT COPY IH LIST		61W0		U108 070
		40H0		U108 080
		10X32		U108 090
		J10		U108 100
TEST IF ARG. IS DETERMINER		10X37		U108 110
		J2		U108 120
		70	92	U108 130
IF NOT, CONTINUE		30H0		U108 140
WITH NEXT ARGUMENT.	93	11W0	90	U108 150
IF A DETERMINER(RECURSE.	92	U108	93	U108 160
NO MORE ARGS., EXIT	91	30H0	J30	U108 170
FIND ON LIST (0) ROUTINE WITH	U110	J42		U110 000
SAME PROCESS(DSCN) AS (1). SET H5		20W0		U110 010
		U111		U110 020
PUT PROC(DSCN) IN W1		20W1		U110 030
COMPARE WITH LIST ITEMS		11W0		U110 040
		1090		U110 050
		J100		U110 060
SET H5 + IF FOUND(- IF NOT		J5	J32	U110 070
STORE ITEM IN W2	90	60W2		U110 080
AND FIND PROC(DSCN)		U111		U110 090
COMPARE IT WITH 1W1		11W1		U110 100
SET H5 - IF FOUND		J2	J5	U110 110
FIND PROCESS OF DSCN	U111	10X20		U111 000
OF ROUTINE (0).		J10		U111 010
(ASSUME IT EXISTS)		10X30	J10	U111 020
TRANSFER H0-LIST ARGUMENT OF	U112	J42		U112 000
ROUTINE (0) TO WS.		60W0		U112 010
BRING IN N-LIST		10X199		U112 020
COPY IT		J73		U112 030
AND STORE IN W1		20W1		U112 040
FIND LIST OF ARGUMENTS		10X20		U112 050
OF DSCN OF (0)		J10		U112 060
		10X31		U112 070
		J10		U112 080
		90		U112 090
		30H0	J32	U112 100
	92	04J0		U112 110
BRING IN TALLY		12W1		U112 112
		10L23		U112 114
		U101		U112 116
		10N1		U112 120
		10N1		U112 121
CONSTRUCT TRANSFER PROCESS		U100		U112 122
INSERT IN J-DEF OF (0)		11W0		U112 130
		J6		U112 140
		10X22	J12	U112 150
LOCATE NEXT ARGUMENT	90	04J60		U112 170
IF NONE, GO TO 92		700		U112 180

	60W2		U112 190
	52H0		U112 200
	10X32		U112 210
FIND TYPE OF ARGUMENT	J10		U112 220
	10X35		U112 230
TEST IF IN H-REGION	J2		U112 240
IF NOT, LOCATE NEXT ARGUMENT	7091		U112 250
IF SO,	12W2		U112 260
COPY OLD ARGUMENT	J74		U112 270
AND MAKE IT LOCAL	J136		U112 275
	11W2		U112 280
	40H0		U112 290
DELETE OLD ARGUMENT	J68		U112 300
	J6		U112 310
	J63		U112 320
INSERT COPY AS NEW ARGUMENT	12W2		U112 330
BRING IN ARGUMENT	40H0		U112 340
	10X36		U112 350
ASSIGN X36 AS NEW TYPE	10X32		U112 360
	J11		U112 370
CHANGE ARG TO WN	U119		U112 380
	11W1		U112 410
	J60		U112 420
	70J7		U112 430
	20W1		U112 440
	92	91	U112 445
	11W2	90	U112 450
LOC. ARGS. IN ROUTINES (2), (1)	U113 J42		U113 000
CORRESPONDING TO HN, N=(0)	20W0		U113 010
IN ROUTINE (1)	U114		U113 020
PUT ARG. LIST OF (1) IN W1	20W1		U113 030
	40H0		
	U125		
	U114		U113 040
PUT ARG. LIST OF (2) IN W2	20W2		U113 050
	11W1	90	U113 060
LOCATE NEXT ARG. OF (1)	J60		U113 070
IF NONE, TERMINATE	7091		U113 075
AND SAVE IT	60W1		U113 080
	11W2		U113 090
LOCATE NEXT ARG. OF (2)	J60		U113 100
	20W2		U113 110
	52H0		U113 120
	10X32		U113 130
	J10		U113 140
TEST IF ARG (1) IN H0	10X35		U113 150
	J2		U113 160
IF NOT, TRY NEXT ARG.	7090		U113 170
TEST IF LOC (ARG)= (0)	12W1		U113 182
	10X37		U113 184
FIND ITS DETERMINER	J10		U113 186
	70J7		U113 188
SET UP TALLY IN W3	J90		U113 190
	J124		U113 192

		40W3		U113	194
		20W3		U113	196
COUNT NUMBER OF X1505	93	J60		U113	202
		7092		U113	204
		12H0		U113	206
		10X150		U113	208
		J2		U113	210
		7093		U113	212
		11W3		U113	214
		J125		U113	216
COMPARE NUMBER WITH (0)	92	30H0	93	U113	218
		30H0		U113	220
		11W3		U113	222
		11W0		U113	224
		J114		U113	226
		30W3		U113	228
IF NOT, TRY NEXT ARG.		7090		U113	230
IF SO, INPUT LOCATION (2)		11W2		U113	240
AND (1), AND EXIT.		11W1	J32	U113	250
TERMINATE	91	30H0	J32	U113	260
FIND ARG. LIST OF DSCN	U114	10X20		U114	000
OF ROUTINE (0)		J10		U114	010
(ASSUMES IT EXISTS)		10X31	J10	U114	020
COMPILE ROUTINE (1) FROM	U115	J43		U115	000
ROUTINE (0). DSCN PROCESSES SAME		J74	W2=R(1)	U115	010
COPY R(0) AND BRING IN TALLY		J136		U115	015
(1) HAS NAMED ARGS.,		10X199	W1=R(0)	U115	020
(0) HAS H-REGION ARGS.		J60	W0=TALLY	U115	030
		J22		U115	040
	91	11W2		U115	050
		11W1		U115	060
		12W0		U115	070
LOCATE ARGS. FOR HN		U113		U115	080
IF NONE, TERMINATE		7090		U115	090
DISCARD ARG. LIST OF R(0)		30H0		U115	100
		52H0		U115	130
COPY SUBSTITUTE ARGUMENT		J74		U115	150
AND MAKE COPY LOCAL		J136		U115	160
		10N1		U115	210
		10N1		U115	220
MAKE IPL WORD = UUWN		U100		U115	230
		11W1		U115	240
		J6		U115	250
INSERT WORD IN JDEF OF R(0)		10X22		U115	260
		J12		U115	270
		11W0		U115	280
ADVANCE TALLY AND EXIT		J60		U115	290
		20W0	91	U115	295
TERMINATE	90	11W2		U115	300
		11W1		U115	310
TRANSFER JDEF OF R(0) TO R(1).		10X22		U115	320
		J10		U115	330
COPY JDEF		J74		U115	335
AND MARK COPY LOCAL		J136		U115	336

		10X22		U115 340
		J11		U115 350
		11W1		U115 360
ERASE COPY OF R(0) AND EXIT		J72	J33	U115 370
COMPILE JDEF OF (0) FROM ROUTINE(1).	U116	40H0		U116 000
BOTH HAVE SAME PROCESS IN DSCN		10X20		U116 010
(0) HAS H-REGION ARGS.		J10		U116 020
(1) HAS W-REGION ARGS.		10X31		U116 030
FIND ARGS. OF DSCN OF (0)		J10		U116 040
		10N0		U116 050
		J120		U116 060
CREATE TALLY IN W0		J50		U116 070
TALLY NO. OF H-ARGS.		1090		U116 080
		J100		U116 090
		J6		U116 100
		10X22		U116 110
		J10		U116 120
COPY J-DEF OF (1)		J74		U116 130
		40H0		U116 140
		J61		U116 150
FIND LAST SYMBOL OF J-DEF		52H0		U116 160
		10X46		U116 170
ERASE ITS LINK		J14		U116 180
SAVE J-DEF		40H0		U116 190
MAKE INSTR. J5N		10N49		U116 200
		91		U116 210
INSERT AT FRONT OF JDEF		J64		U116 220
		40H0		U116 230
		10N29		U116 240
MAKE INSTR. J3N		91		U116 250
		40H0		U116 260
ASSIGN NO AS LINK OF INSTRUCTION		92		U116 280
INSERT INSTR. AT END OF JDEF		J65		U116 300
		10X22		U116 310
INSERT JDEF IN ROUT. (0).		J11	J30	U116 320
	90	14X32		U116 330
TEST IF ARGUMENT IS		J10		U116 340
IN H-REGION		10X35		U116 350
		J2		U116 360
IF NOT, CONTINUE		70J4		U116 370
IF SO, TALLY AND CONTINUE		11W0		U116 380
		J125	J8	U116 390
	91	4J0		U116 400
		11W0		U116 410
CONSTRUCT NEW INSTRUCTION		J120		U116 420
		40H0		U116 430
		J110		U116 440
		10L10		U116 450
		U101	U106	U116 460
ASSIGN NO AS LINK OF INSTR.	92	4J90		U116 470
CREATE LOCAL SYMBOL		J136		U116 480
		40H0		U116 490
SAVE IT		J50		U116 500
BRING IN NO		10N0		U116 510

		10X33		U116	520
MAKE IT REGION OF SYMBOL		J11		U116	530
		11W0		U116	540
MAKE 1W0 LINK OF INSTR.		10X46		U116	550
		J11	J30	U116	560
REPLACE HS BY WS IN DSCN	U117	J90		U117	000
OF (0). OUTPUT (0)=N, NUMBER		J124		U117	010
OF REPLACEMENTS.		J50		1W0=TALLY	U117 020
		10X20		U117	030
		J10		U117	040
FIND LIST OF ARGS OF DSCN OF (0)		10X31		U117	050
		J10		U117	060
DO 90 TO ARGS ON LIST		1090		U117	070
		J100		U117	080
BRING IN TALLY AND EXIT		11W0	J30	U117	090
SAVE ARG.	90	44H0		U117	100
		10X32		U117	110
FIND ITS TYPE		J10		U117	120
		10X37		U117	130
TEST IF DETERMINER		J2		U117	140
		70	91	U117	150
		40H0		U117	155
DESIG. SYMB. FIND REGION		10X45		U117	160
		J10		U117	170
		10X33		U117	190
		J10		U117	200
TEST IF H-REGION		10L8		U117	210
		J2		U117	220
IF NOT, EXIT		7092		U117	230
CHANGE ARG TO WN		U119		U117	240
AND ADD ONE TO TALLY		11W0		U117	270
		J125		U117	280
	92	30H0	J4	U117	290
DETERMINER.	91	10X20		U117	300
FIND ITS LIST OF ARGUMENTS		J10		U117	310
		10X31		U117	320
		J10		U117	330
		1090		U117	340
		J100	0	U117	350
MODIFY JDEF OF (0) TO TAKE N=(1)	U118	J51		U118	000
INPUTS FROM WS. NO OUTPUT.		11W0		U118	010
		10X22		U118	020
		J10		U118	030
FIND AND SAVE JDEF OF (0)		40H0		U118	040
FIND ITS LAST SYMBOL		J61		U118	050
		52H0		U118	060
		10X46		U118	070
ERASE ITS LINK		J14		U118	080
SAVE JDEF		40H0		U118	090
MAKE INSTR. J5N		10N49		U118	100
		91		U118	110
INSERT AT FRONT OF JDEF		J64		U118	120
MAKE INSTR. J3N		10N29		U118	140
		91		U118	150

ASSIGN NO AS LINK OF INSTR.		40H0		U118 160
INSERT INSTR. AT END OF JDEF		92		U118 170
CONSTRUCT NEW INSTRUCTION	91	J65	J31	U118 180
BRING IN N		4J0		U118 190
COMPUTE XN		11W1		U118 200
		J120		U118 210
		40H0		U118 220
		J110		U118 230
MAKE JXN		10L10		U118 240
		U101	U106	U118 250
ASSIGN NO AS LINK OF INSTR.	92	4J90		U118 260
CREATE LOCAL SYMBOL		J136		U118 270
		40H0		U118 280
SAVE IT		J50		U118 290
BRING IN NO		10N0		U118 300
		10X33		U118 310
MAKE IT REGION OF SYMBOL		J11		U118 320
MAKE SYMBOL LINK OF INSTR.		11W0		U118 330
		10X46		U118 340
		J11	J30	U118 350
CHANGE ARG (0)=HN TO WN,	U119	40H0		U119 000
NO OUTPUT		10X37		U119 010
FIND ITS DETERMINER		J10		U119 020
SET UP TALLY IN WC		J90		U119 040
		J124		U119 050
		J50		U119 060
TALLY AND ERASE X1505	93	40H0		U119 070
FROM DETERMINER		10X150		U119 080
		J62		U119 090
		7094		U119 100
		11W0		U119 110
		J125		U119 120
		30H0		U119 130
		J68	93	U119 140
CONSTRUCT SYMBOL WN	94	30H0		U119 150
		30H0		U119 160
		11W0		U119 170
		30W0		U119 180
		10L23		U119 190
		U101		U119 200
ASSIGN IT AS X45 OF ARGUMENT		10X45		U119 210
		J11	0	U119 220
MODIFY JDEF TO DO PROCESS (2)	U120	10N9		U120 005
AFTER (3) WITH FIELD (1)=(0). SET		U101		U120 010
H5+ IF (2) WAS NAMELESS, H5- IF		J53		U120 015
(2) HAS NAME. NO OUTPUT.		11W2		U120 020
		J81		U120 025
FIND FIRST INSTR. OF (1)		40H0		U120 030
		10X41		U120 040
IF IT HAS NAME, REPLACE (0)		J10		U120 050
WITH NAME, IF NOT, SKIP TO 90.		7090		U120 060
		20W0		U120 070
		30H0	91	U120 080
PUT (0) AS NAME OF FIRST OF (1).	90	11W0		U120 090

		10X41		U120 100
		J11		U120 110
FIND LAST INSTR. OF (3)	91	11W3		U120 120
		J61		U120 130
		52H0		U120 140
MAKE TWO ITS LINK.		11W0		U120 150
OR SYMB		11W1		U120 160
		J11		U120 170
SET H5 AND EXIT		J5	J33	U120 180
MODIFY JDEF TO EXIT AFTER	U122	J61		U122 000
PROCESS (0). NO OUTPUT		52H0		U122 010
		40H0		U122 012
		10X46		U122 014
		J10		U122 016
		70	90	U122 018
MAKE SYMB. 0.		10N0		U122 020
		40H0		U122 030
		U101		U122 040
ASSIGN IT AS LINK OF LAST OF (0).		10X46	J11	U122 050
	90	30H0	J8	U122 060
PRINT THE JDEF OF (0)	U125	10X22		U125 000
FIND JDEF		J10		U125 010
CLEAR PRINT LINE		J154		U125 020
PRINT LIST OF INSTRUCTIONS		1090		U125 030
		J100	E50	U125 040
PROCESS INSTRUCTION	90	04J0		U125 050
		10X40		U125 055
		10N28		U125 060
ENTER TYPE IN COL. 28		920		U125 070
		10X41		U125 080
		10N30		U125 090
ENTER NAME IN COLS. 30-34		910		U125 100
		10X42		U125 110
		10N35		U125 120
ENTER SIGN IN COL. 35		920		U125 130
		10X43		U125 140
		10N36		U125 150
ENTER P IN COL. 36		920		U125 160
		10X44		U125 170
		10N37		U125 180
ENTER Q IN COL. 37		920		U125 190
		10X45		U125 200
		10N38		U125 210
ENTER SYMB IN COLS. 38-42		910		U125 220
		10X46		U125 230
		10N44		U125 240
ENTER LINK IN COLS. 44-48.		910		U125 245
PRINT.		E50		U125 250
		30H0	J4	U125 255
FIND VALUE OF ATTRIBUTE (1)	910	04930		U125 260
AND ENTER ITS REGION AND		70J31		U125 270
LOCATION IN COLS. BEGINNING		J160		U125 280
		40H0		U125 285
AT (0).		10X33		U125 290

ENTER REGION		J10		U125 300
		70912		U125 310
		J157		U125 320
	911	10X34		U125 330
ENTER LOCATION		J10		U125 340
		70J31		U125 350
		J157	J31	U125 360
	912	10N1		U125 365
		J161	911	U125 367
FIND VALUE OF ATTRIBUTE (1)	920	04930		U125 370
AND ENTER IN COL. (0)		70J31		U125 380
		J160		U125 390
		J157	J31	U125 400
PROCESS FOR 910 AND 920	930	04J51		U125 410
		40H0		U125 420
		11W1		U125 430
		J10		U125 440
		700		U125 450
		11W0	0	U125 460
ENTER IN PRINT LINE THE NAME OF	U126	J41		U126 000
		40H0		U126 001
TEST IF ROUTINE		10X32		U126 002
IS A DETERMINER		J10		U126 003
		7096		U126 004
		10X37		U126 005
		J2		U126 006
IF NOT, GO TO 96		7096		U126 007
IF SO, PRINT ITS VERB		40H0		U126 008
		10X20		U126 009
		J10		U126 010
ENTER DATA TERMS OF VERB	98	J60		U126 011
		7097		U126 012
		12H0		U126 013
		910	9b	U126 014
	97	E51		U126 015
AND CONTINUE AT 96		30H0	96	U126 016
	96	10X20		U126 019
FIND ITS DSCN		J10		U126 020
		40H0		U126 030
		10X30		U126 040
PUT PROCESS IN W0		J10		U126 050
		20W0		U126 060
		10X31		U126 070
		J10		U126 080
PUT ARGS IN W1		20W1		U126 090
	91	11W0		U126 100
FIND NEXT WORD IN NAME		J60		U126 110
EXIT IF NONE		7092		U126 120
		60W0		U126 130
		52H0		U126 140
		40H0		U126 150
TEST IF 1H0 NAMES DATA TERM		J131		U126 160
		7090		U126 170
		910	91	U126 180

DOES NOT NAME DATA TERM	90	30H0		U126 190
		11W1		U126 191
SPACE ONCE		E51		U126 195
FIND NEXT ARGUMENT		J60		U126 200
ERROR STOP		70J7		U126 210
		60W1		U126 220
		52H0		U126 230
		40H0		U126 240
		10X32		U126 250
		J10		U126 260
		10X37		U126 270
TEST IF ARG IS A DETERMINER		J2		U126 280
		70	94	U126 290
IF NOT, SAVE IT		40H0		U126 300
TEST IF MEMBER OF H0 LIST		10X32		U126 301
		J10		U126 302
		10X35		U126 303
		J2		U126 304
IF SO, BRANCH TO 95		70	95	U126 305
IF NOT, SAVE IT		40H0		U126 306
FIND ITS DETERMINER		10X37		U126 310
		J10		U126 315
		1093		U126 320
AND ENTER IT		J100		U126 330
SPACE ONCE		E51		U126 335
FIND ITS SYMBOL		10X45		U126 340
		J10		U126 350
		40H0		U126 360
		10X33		U126 370
		J10		U126 380
		910		U126 390
		10X34		U126 400
		J10		U126 410
		910		U126 420
		E51	91	U126 425
	92	30H0	J31	U126 430
	93	10910		U126 440
		J100	0	U126 450
	94	U126	91	U126 460
ENTER IN PRINT LINE	910	40H0		U126 470
		J157		U126 475
LINE SPACE IF NO ROOM		70	J8	U126 480
		E50		U126 490
		10N10		U126 500
		J160	910	U126 510
ENTER NAME OF MEMBER	95	J60		U126 530
OF H0 LIST		70J7		U126 540
		52H0		U126 550
		910		U126 560
		E51	91	U126 570
PRINT THE SDSC OF ROUTINE (0)	U127	J40		U127 000
CLEAR PRINT LINE		J154		U127 010
		10X24		U127 020
FIND SDSC		J10		U127 030

FIND LIST OF AFFECTED CELLS		70J30		U127 040
		10X71		U127 050
		J10		U127 060
DO 90 TO AFFECTED CELLS		70J30		U127 070
AND EXIT		1090		U127 080
		J100	J30	U127 090
	90	60W0		U127 100
FIND NAME OF CELL		10X41		U127 110
		J10		U127 120
ENTER IT IN PRINT LINE		91		U127 130
		E53		U127 140
		11W0		U127 150
FIND INPUT LIST		10X75		U127 160
		J10		U127 170
AND ENTER ITS SYMBOLS		1092		U127 180
		J100		U127 190
ENTER PERIOD		E53		U127 200
		11W0		U127 210
FIND OUTPUT LIST		10X76		U127 220
		J10		U127 230
ENTER ITS SYMBOLS		1092		U127 240
		J100		U127 250
ENTER PERIOD AND PRINT LINE		E53	E50	U127 260
ENTER A SYMBOL IN THE PRINT LINE	91	40H0		U127 270
		10X33		U127 280
		J10		U127 290
		J157		U127 300
		10X34		U127 310
		J10	J157	U127 320
ENTER THE SYMBOLS, THEN COMMAS	92	91	E54	U127 330
PRINT NAME, DSCN, JDEF, AND	U128	J50		U128 000
SDSC OF ROUTINE (0)		90		U128 010
		E50		U128 020
		11W0		U128 040
		40H0		U128 050
		10X20		U128 060
TEST OF IT HAS A DSCN		J10		U128 070
IF NOT, SKIP TO 92		7092		U128 080
		30H0		U128 090
ENTER DSCN IN PRINT LINE		U126		U128 100
PRINT		E50		U128 110
AND SPACE		E50		U128 120
		11W0		U128 130
	92	40H0		U128 140
		10X22		U128 150
TEST IF IT HAS A JDEF		J10		U128 160
IF NOT, SKIP TO 92		7093		U128 170
		30H0		U128 180
PRINT JDEF		U125		U128 190
AND SPACE		E50		U128 200
		11W0		U128 210
	93	40H0		U128 220
		10X24		U128 230
TEST IF IT HAS A SDSC		J10		U128 240

IF NOT, EXIT BY 95		7095		U128	250
		30H0		U128	290
PRINT SDSC AND EXIT		U127	J30	U128	300
	95	30H0	J30	U128	310
	90	04J0		U128	320
		11W0		U128	330
TEST IF IT HAS AN IPLN		10X25		U128	340
		J10		U128	350
IF NOT, EXIT		700		U128	360
		40H0		U128	370
IF SO, ENTER IN PRINT LINE		10X33		U128	380
		J10		U128	390
		J157		U128	400
		10X34		U128	410
		J10	J157	U128	420
COMPILE JDEF OF ROUT.(0) FROM	U130	J43		U130	000
JDEF OF ROUT.(1). ROUT(1) IS A J.		J21		U130	010
		11W0		U130	020
COPY R1W0 AND		J74		U130	030
COPY ARGS OF DSCN		40H0		U130	031
		U108		U130	040
		60W2		U130	043
		U117		U130	050
REPLACE HS BY WS IN DSCN OF COPY.		20W3		U130	060
SAVE(0)=NUMBER OF REPLACEMENTS.		11W1		U130	070
		11W2		U130	080
COMPILE JDEF OF 1W2		U131		U130	090
FROM JDEF OF 1W1.		11W3		U130	100
MODIFY JDEF OF 1W2 TO TAKE		11W2		U130	110
N=1W3 INPUTS FROM WS.		U118		U130	120
		11W0		U130	130
BRING IN 1W0.		11W2		U130	140
		10X22		U130	150
FIND JDEF OF 1W2		J10		U130	160
AND ASSIGN IT AS JDEF OF 1W0		J74		U130	161
		J136		U130	162
		10X22		U130	165
		J11		U130	170
		910		U130	175
ASSIGN NAME TO JDEF.		11W2		U130	180
		J72		U130	190
ERASE 1W2,		11W3		U130	200
AND TALLY CELL,		J9	J33	U130	210
AND EXIT	910	11W0		U130	220
		10X22		U130	230
FIND JDEF		J10		U130	240
		J81		U130	250
FIND FIRST INSTRUCTION.		40H0		U130	260
		10X41		U130	265
		J10		U130	266
IF IT HAS NO NAME,		70	911	U130	270
		11W0		U130	280
		10X25		U130	290
ASSIGN IT V(X25) OF ROUTINE.		J10		U130	300

IF IT HAS NAME, EXIT.	911	10X41	J11	U130	310
COMPILE JDEF OF ROUT.(0)		30H0	J8	U130	320
FROM JDEF OF ROUT.(1), (0) IS	U131	J45		1W0=(0)	U131 000
A J, (1) HAS W-REGION INPUTS		J21		1W1=(1)	U131 010
		J90			U131 020
		J124			U131 030
SET UP TALLY IN W2		20W2		1W2=TALLY	U131 040
		11W0			U131 050
		11W1			U131 060
COPY JDEF OF (1) AND		10X22			U131 070
ASSIGN COPY AS JDEF OF (0).		J10			U131 080
		J74			U131 090
		J136			U131 100
		10X22			U131 110
		J11			U131 120
LOCATE ARGS IN 1W0 AND 1W1	90	11W0			U131 130
FOR HN, N=TALLY, IN 1W1,		11W1			U131 140
		11W2			U131 150
		U113			U131 160
IF NONE, EXIT.		7091			U131 170
SAVE ARG. OF 1W1 IN W3.		52H0			U131 180
		J74			U131 190
		20W3			U131 200
		11W2			U131 210
ADD 1 TO TALLY		J125			U131 220
		30H0			U131 230
SAVE LOC. OF ARG OF 1W0 IN W4		60W4			U131 240
FIND ARG. OF 1W0		52H0			U131 250
COMPILE (FIND ARG.)		U132			U131 260
		11W0			U131 270
INSERT (FIND ARG.) AT FRONT		10X22			U131 280
OF JDEF OF 1W0		J10			U131 290
		92			U131 300
REPLACE ARG OF 1W0 WITH		11W3			U131 310
ARG OF 1W1		21W4	90		U131 320
	91	11W2			U131 330
ERASE TALLY AND EXIT		J9	J35		U131 340
INSERT COMPILED (FIND ARG.)	92	44H0			U131 350
DIVIDE JDEF		J75			U131 360
SAVE REMAINDER		20W5			U131 370
		J6			U131 380
ADD (FIND ARG.) TO HEAD		J76			U131 390
ADD REMAINDER		11W5			U131 400
		J76	J8		U131 410
COMPILE (FIND ARG) (0). FINAL	U132	40H0			U132 000
ARGS. ARE DESIG. SYMBS. OUTPUT (0)		J90			U132 010
IS LIST OF INSTRS. WITHOUT TERM.		J136			U132 020
		40W1			U132 030
CREATE OUTPUT LIST IN W1.		20W1			U132 040
		10X32			U132 050
TEST IF ARGUMENT		J10			U132 060
IS DETERMINER.		10X37			U132 070
		J2			U132 080
ARGUMENT NOT A DETERMINER,		70	90		U132 090

		J90	U132 100
SET UP Q-TALLY IN W0.		J124	U132 110
		J50	U132 120
SAVE ARGUMENT	92	40H0	U132 130
FIND ITS DETERMINER		10X37	U132 135
		J10	U132 140
	94	J60	U132 145
		7093	U132 150
TALLY X151S=Q		12H0	U132 155
OF DETERMINER IN W0		10X151	U132 160
		J2	U132 170
		7094	U132 180
		11W0	U132 190
		J125	U132 200
COPY SYMBOL OF ARGUMENT	93	30H0	U132 210
		30H0	U132 220
		10X45	U132 225
		J10	U132 230
		J74	U132 240
MAKE INSTRUCTION IQS.		11W0	U132 250
		10N1	U132 260
		U100	U132 270
INSERT INSTR. AT FRONT		11W1	U132 300
OF LIST 1W1.		J6	U132 310
EXIT, LEAVING 1W1.		J64	U132 320
		11W1	U132 330
SET UP INSTR. SUBLIST,	90	J90	U132 340
AND MARK IT LOCAL.		J136	U132 350
AND SAVE IT IN W0.		J50	U132 360
SAVE DETERMINER		40H0	U132 363
FIND ITS DSCN		10X20	U132 365
		J10	U132 366
FIND LIST OF ARGUMENTS		10X31	U132 380
AND PROCESS THEM.		J10	U132 390
		10910	U132 400
		J100	U132 410
FIND JDEF		10X22	U132 420
		J10	U132 430
COPY IT		J74	U132 432
		J136	U132 433
		40H0	U132 434
FIND LAST INSTRUCTION		J61	U132 435
		52H0	U132 436
ERASE ITS LINK		10X46	U132 437
		J14	U132 438
		11W0	U132 440
FIND END OF LIST		J61	U132 445
INSERT LIST ON MAIN LIST		95	U132 460
INSERT SUBLIST IN LIST		11W0	U132 470
		11W1	U132 480
		95	U132 490
INSERT LIST ON MAIN LIST		11W1	U132 500
BRING IN LIST AND EXIT.	910	U132	U132 510
COMPILE (FIND ARG.)		11W0	U132 520
AND INSERT IN SUBLIST.			

INSERT LIST ON MAIN LIST		95	J4	U132	530
INSERT LIST AT FRONT OF MAIN LIST	95	44H0		U132	540
		J75		U132	550
		40W2		U132	560
		20W2		U132	570
		J6		U132	580
		J76		U132	590
		11W2		U132	600
		J76		U132	610
		30W2	J8	U132	620
	U133	J45		U133	000
COMPILE JDEF WITH NAME (2)		10X199		1W4=NAME	U133 005
LIST OF SEGMENTS (1) AND FLOW		J73		1W3=SEG L	U133 010
CHART (0). OUTPUT (0) IS JDEF		J136		1W2=FLOW	U133 020
NAME NEW JDEF		J90		1W1=TALLY	U133 030
		J136		1W0=JDEF	U133 040
		J24		U133	050
ASSIGN NAME TO FIRST INSTRUCTION		11W3		U133	053
		J81		U133	054
		J81		U133	055
		11W4		U133	056
		10X41		U133	057
		J11		U133	058
		11W2		U133	060
		40W3		U133	070
ATTACH SEGMENTS TO FLOW CHART	91	J60		U133	080
		7090		U133	090
		12H0		U133	100
		11W3		U133	110
		J60		U133	120
		70J7		U133	130
		60W3		U133	140
		52H0		U133	145
		10X22		U133	150
		J11	91	U133	160
ASSIGN ADDRESSES TO SEGMENTS	90	30H0		U133	170
		30W3		U133	180
		11W2		U133	190
		1092		U133	200
		J100		U133	210
		11W0		U133	215
ASSEMBLE SEGMENTS IN NEW JDEF		11W3		U133	220
		10910		U133	230
		J100		U133	235
		40H0		U133	240
		U122	J35	U133	245
	910	11W0		U133	251
		J6		U133	260
		J76		U133	262
		20W0	J4	U133	264
ASSIGN ADDRESSES TO SEGMENT	92	64W5		1W5=SEGM	U133 270
FIND JDEF OF SEGMENT		10X22		U133	280
		J10		U133	290
		11W5		U133	310

END P OF SEGMENT	10X43		U133 320
	J10		U133 330
TEST IF P=7	10N7		U133 340
	J2		U133 350
IF NOT, GO TO 93	7093		U133 360
IF P=7	11W5		U133 370
	40H0		U133 375
	925		U133 376
FIND SYMB OF SEGM	10X45		U133 380
	J10		U133 390
	40H0		U133 400
TEST IF SYMB IS 0	10N0		U133 410
	J2		U133 420
IF NOT, GO TO 94	7094		U133 430
IF SO, GO TO 93	30H0	93	U133 440
	94 11W2		U133 450
	J6		U133 460
FIND SYMB ON FLOW	920		U133 470
	10X22		U133 480
FIND ITS JDEF	J10		U133 490
	70J7		U133 500
BRING IN (FIELD)=(SYMB)	10X45		U133 510
	11W1		U133 520
	J60		U133 530
	60W1		U133 540
	52H0		U133 545
LINK SEGM TO SYMB	U120		U133 550
	11W5		U133 560
	10X22		U133 570
BRING IN JDEF OF SEGM	J10		U133 580
FIND LINK OF SEGM	93 11W5		U133 590
	10X46		U133 600
	J10		U133 610
	40H0		U133 620
TEST IF LINK IS 0	10N0		U133 630
	J2		U133 640
	7095		U133 650
IF IT IS, EXIT	30H0	J8	U133 660
IF LINK IS NOT 0	95 11W2		U133 670
	J6		U133 680
FIND SYMB ON FLOW	920		U133 690
	10X22		U133 700
FIND ITS JDEF	J10		U133 710
	70J7		U133 720
BRING IN (FIELD)=(LINK)	10X46		U133 730
	11W1		U133 740
	J60		U133 750
	60W1		U133 755
	52H0		U133 756
LINK SEGM TO SYMB	U120	J4	U133 760
FIND SYMB ON FLOW	920 04J50		U133 765
	10921		U133 770
	J100	J30	U133 775
FIND NAME OF SYMB	921 40H0		U133 780

COMPARE WITH 1W0		10X41		U133 785
		J10		U133 790
		11W0		U133 795
		J2		U133 800
EXIT, STOP GENERATOR		70	J3	U133 805
EXIT, CONTINUE GENERATOR		30H0	J4	U133 810
	925	14X22		U133 815
		J10		U133 820
		J90		U133 825
		J136		U133 830
		40H0		U133 835
		10N7		U133 840
		10X43		U133 845
		J11	J65	U133 850
COMPILE (0) FROM ITS DSCN	U134	J50		U134 000
FIND LIST OF COMPILED ROUTINES		10X196		U134 010
AND SEEK SOURCE ROUTINE		1090		U134 020
		J100		U134 030
		70	J30	U134 040
IF FOUND, COMPILE (0)		11W0		U134 050
FROM DSCN OF XOURCE		U130	J30	U134 060
FIND SOURCE ROUTINE FOR (0)	90	40H0		U134 070
		10X20		U134 080
FIND PROCESS OF DSCN		J10		U134 090
		7091		U134 095
OF SOURCE		10X30		U134 100
		J10		U134 110
		7091		U134 115
		11W0		U134 120
		10X20		U134 130
		J10		U134 140
		10X30		U134 150
		J10		U134 160
COMPARE WITH PROCESS OF (0)		J2		U134 170
EXIT WITH SIGNAL		70	J3	U134 180
TO GENERATOR	91	30H0	J4	U134 190
COMPILE JDEF OF ROUTINE (0).	U135	J41		U135 000
OUT.(0) IS JDEF, IF EXISTS. SET H5.		60W0		U135 010
TEST IF IT EXISTS	90	10X22		U135 020
		J10		U135 030
IF SO, EXIT WITH H5+		70	J31	U135 040
		11W0		U135 050
FIND CLOSEST DEFINITION		U136		U135 060
IF NONE, EXIT WITH H5-		70J31		U135 070
		40H0		U135 080
FIND AND APPLY		11W0		U135 090
RELEVANT PROCESS		U137		U135 100
		11W0		U135 110
FIND CLOSEST DEFINITION		U136		U135 120
TEST PROGRESS		U138		U135 140
EXIT OR REPEAT.		70J31		U135 150
		11W0	90	U135 160
FIND CLOSEST DEF. OF ROUTINE (0)	U136	J50		U136 000
		11W0		U136 010

		10X20		U136 020
		J10		U136 030
		70	90	U136 040
		11W0		U136 050
		10X24		U136 060
		J10		U136 070
		70J30	91	U136 080
	90	30H0		U136 090
		10X20	J30	U136 100
	91	30H0		U136 110
		10X24	J30	U136 120
FIND AND APPLY RELEVANT PROCESS	U137	J6		U137 000
TO ROUTINE (0) WITH DEF. (1)		10X198		U137 010
NO OUTPUT		J6		U137 020
		J10		U137 030
		J1	0	U137 040
TEST IF DEF (0) IS CLOSER TO	U138	10X197		U138 000
JDEF THAN DEF (1)		J6		U138 010
		J62		U138 020
		J6		U138 030
		J62	0	U138 040
COMPOSE FLOW CHART OF ROUTINE (0)	U139	J42		U139 000
FROM ITS JDEF		60W0		U139 010
		J90		U139 020
		J136		U139 030
STORE NAME OF CHART IN W1		60W1		U139 040
		10X26		U139 050
ASSIGN IT TO ROUTINE 1W0		J11		U139 060
		11W0		U139 070
		10X22		U139 080
FIND JDEF OF ROUTINE		J10		U139 090
COPY JDEF, MARK IT LOCAL		J74		U139 100
		J136		U139 110
STORE IT IN W2	91	60W2		U139 120
		J90		U139 130
		J136		U139 140
CREATE SEGMENT		40H0		U139 150
AND STORE IT		11W1		U139 160
		J6		U139 170
INSERT SEGMENT		J65		U139 180
AT END OF FLOW CHART		J6		U139 190
MAKE JDEF THE		10X22		U139 200
X22 OF SEGM.		J11		U139 210
		11W2		U139 220
FIND NEXT INSTRUCTION	92	J60		U139 230
		7094		U139 240
		12H0		U139 260
FIND ITS P		10X43		U139 270
		J10		U139 280
		7090		U139 285
TEST IF P=7		10N7		U139 290
		J2		U139 300
IF SO, DIVIDE SEGMENT		7090	93	U139 310
AND REPEAT.	93	J75	91	U139 320

IF NOT,	90	12H0		U139 330
		10X46		U139 340
		J10		U139 350
TEST IF LINK IS LOCAL		7092		U139 360
IF SO,		10X33		U139 370
DIVIDE SEGMENT AND REPEAT.		J10		U139 380
		10N9		U139 390
		J2		U139 400
IF NOT, FIND NEXT INSTRUCTION		7092	93	U139 410
	94	30H0		U139 420
		11W1		U139 430
LOCATE NEXT SEGMENT	98	J60		U139 440
IF NONE, EXIT		7099		U139 450
		12H0		U139 460
PUT ITS NAME IN W0		60W0		U139 470
		10X22		U139 480
FIND ITS JDEF		J10		U139 490
AND PUT IT IN W2		60W2		U139 500
FIND FIRST INSTRUCTION		J81		U139 510
		10X41		U139 520
FIND ITS NAME		J10		U139 530
IF NONE, SKIP TO 95		7095		U139 540
		11W0		U139 550
		J6		U139 560
ASSIGN IT AS SEGMENT NAME		10X41		U139 570
		J11		U139 580
	95	11W2		U139 590
FIND LAST INSTRUCTION		J61		U139 600
		52H0		U139 610
STORE IT IN W2		60W2		U139 620
		10X43		U139 630
TEST IF ITS P=7		J10		U139 640
		7096		U139 645
		10N7		U139 650
		J2		U139 660
IF NOT, SKIP TO 96		7096		U139 670
IF SO, SET P=7 IN SEGMENT		11W0		U139 680
		10N7		U139 690
		10X43		U139 700
		J11		U139 710
ASSIGN SYMB. OF INSTRUCTION		11W0		U139 720
TO SEGMENT		11W2		U139 730
		10X45		U139 740
		J10		U139 750
		10X45		U139 760
		J11	97	U139 770
SET P=0 IN SEGMENT	96	11W0		U139 780
		10N0		U139 790
		10X43		U139 800
		J11	97	U139 810
ASSIGN LINK OF INSTRUCTION	97	11W0		U139 820
TO SEGMENT		11W2		U139 830
		10X46		U139 840
		J10		U139 850

		10X46		U139 860
		J11	98	U139 870
EXIT	99	30H0	J32	U139 880
COMPILE JDEF OF ROUTINE (0)	U140	J50		U140 000
FROM ITS SDSC. NO OUTPUT		11W0		U140 001
COMPILE IT		930		U140 003
ASSIGN A NAME TO IT		910		U140 005
ASSIGN A TERMINATION TO IT		920	J30	U140 007
	930	J42		U140 009
FROM ITS SDSC. NO OUTPUT.		60W0	1W0=(0)	U140 010
		10X24		U140 020
FIND SDSC.		J10		U140 030
FIND INPUT-OUTPUT DIFFERENCE.		X7		U140 040
		40H0		U140 050
		10X50		U140 055
		J10		U140 056
TEST FOR NO DIFFERENCE.		10X80		U140 060
		J2		U140 070
		7090		U140 080
		J32		U140 090
IF NO DIFFERENCE		J8	J3	U140 095
EXIT WITH H5=-				U140 100
IF DIFFERENCE EXISTS,	90	11W0		U140 110
PRODUCE NEW ROUTINE WITHOUT DIFF.,		U141		U140 120
STORE IT IN W1.		60W1		U140 130
CONSTRUCT SDSC FOR A NEW ROUTINE,		11W0		U140 140
I(1W0) I(1W1).		U144		U140 150
SAVE NEW ROUTINE		60W2		U140 160
COMPILE ITS JDEF		930		U140 165
IF NULL, SKIP TO 91		7091		U140 170
BRING IN NEW ROUTINE		11W2		U140 180
PREFIX JDEF OF NEW ROUTINE TO JDEF		11W1		U140 190
OF 1W1, AND MODIFY SDSC OF 1W1.		U145		U140 195
	91	11W2		U140 200
ERASE PREFIXED ROUTINE		J72		U140 210
		11W0		U140 220
CONSTRUCT SDSC FOR A NEW ROUTINE,		11W1		U140 230
O(1W1) O(1W0).		U146		U140 240
SAVE NEW ROUTINE		60W2		U140 250
COMPILE ITS JDEF.		930		U140 255
IF NULL, SKIP TO 92		7092		U140 260
BRING IN NEW ROUTINE		11W2		U140 270
SUFFIX JDEF OF NEW ROUTINE TO JDEF		11W1		U140 280
OF 1W1, AND MODIFY SDSC OF 1W1.		U147		U140 285
ERASE SUFFIXED ROUTINE	92	11W2		U140 290
		J72		U140 300
BRING IN 1W0.		11W0		U140 310
FIND JDEF OF 1W1,		11W1		U140 320
		10X22		U140 330
		J10		U140 340
COPY IT.		J74		U140 350
		J136		U140 360
ASSIGN COPY AS JDEF OF 1W0.		10X22		U140 370
		J11		U140 380
		11W1		

ERASE 1W1		J72		U140	390
AND EXIT WITH H5=+		J32	J4	U140	400
	910	11W0		U140	410
FIND JDEF.		10X22		U140	420
		J10		U140	430
FIND FIRST INSTRUCTION.		J81		U140	440
		40H0		U140	450
		10X41		U140	455
		J10		U140	456
IF IT HAS NO NAME,		70	911	U140	460
		11W0		U140	470
ASSIGN IT V(X25) OF ROUTINE.		10X25		U140	480
		J10		U140	490
		10X41	J11	U140	500
IF IT HAS NAME, EXIT.	911	30H0	J8	U140	510
	920	11W0		U140	520
		10X22		U140	530
		J10		U140	540
		40H0		U140	550
TEST IF LAST INSTRUCTION		J61		U140	560
HAS A LINK.		52H0		U140	570
		10X46		U140	580
		J10		U140	590
IF NOT, ASSIGN LINK 0		70U122		U140	600
IF SO, EXIT.		30H0	J8	U140	610
PRODUCE ROUTINE FROM (0)	U141	J44		(0)=1W0	U141 000
WITHOUT DIFFERENCE (1).		J21		(1)=1W1	U141 010
OUTPUT (0) IS NEW ROUTINE.		11W1			U141 020
FIND ROUTINE RELEVANT TO DIFF.		U150			U141 030
		J74			U141 040
COPY IT AND MARK COPY LOCAL.		J136			U141 050
PUT RELEVANT ROUT. IN W2		60W2			U141 060
		10X24			U141 070
		J10			U141 080
PUT ITS SDSC IN W3		60W3			U141 090
SAVE IT		40H0			U141 100
LOCATE ITS VARIABLE CELL.		U152			U141 110
IF NONE, GO TO 90, AFTER J8		7093			U141 120
FIND ITS NAME		52H0			U141 130
FIND ITS VARIABLES		10X41			U141 135
		J10			U141 136
		11W1			U141 140
		10X41			U141 150
FIND VARIABLE OF DIFFERENCE,		J10			U141 160
SUBSTITUTE IT FOR VAR. CELL NAME.		U153			U141 170
BRING IN SDSC OF REL. ROUT.	90	11W3			U141 180
FIND VARIABLE OF DIFF.		11W1			U141 190
		10X41			U141 200
		J10			U141 210
LOCATE CELL OF SDSC WITH THIS NAME		U151			U141 220
IF NONE, GO TO 91.		7091			U141 230
		52H0			U141 232
		10X41			U141 234
		J10			U141 235

BRING IN SDSC OF REL. ROUT.		11W3		U141	240
		11W0		U141	250
FIND SDSC OF 1W0		10X24		U141	260
		J10		U141	270
SUB. NAMES OF (0) FOR VAR. OF W3		U154		U141	280
	91	11W0		U141	290
		10X24		U141	300
		J10		U141	310
FIND LIST OF AFFECTED CELLS OF 1W0		10X71		U141	320
		J10		U141	330
LOCATE NEXT CELL	92	J60		U141	340
		7094		U141	350
FIND CELL		12H0		U141	360
FIND ITS NAME		10X41		U141	370
		J10		U141	380
STORE IT IN W4		60W4		U141	390
FIND W3		11W3		U141	400
		J6		U141	410
LOCATE CELL OF W3 WITH SAME NAME		U151		U141	420
IF NONE, FIND NEXT.		7092		U141	430
		30H0		U141	440
		11W4		U141	450
		11W3		U141	460
SUBSTITUTE NAMES OF SDSC (W0) FOR		11W0		U141	470
VARIABLES OF (W3) IN CELL (W4)		10X24		U141	480
		J10		U141	490
		U154	92	U141	500
	95	11W1		U141	510
FIND ROUTINE RELEVANT TO W1,		U150		U141	520
FIND MODIFIED COPY.		11W2		U141	530
MAKE JDEF OF (0) FROM (1).		U155		U141	540
LEAVE NEW ROUTINE		11W2	J34	U141	550
	93	30H0		U141	560
		30H0	90	U141	565
	94	30H0	95	U141	570
CONSTRUCT SDSC FOR A NEW ROUTINE	U142	J53		U142	000
(0) OF (1) TO (2) OF (3). (0)=NAME		11W1		U142	010
		10X24		U142	015
		J10		U142	016
OF NEW ROUTINE.		10X71		U142	020
		J10		U142	030
COPY X71 OF 1W1		J74		U142	035
		J136		U142	036
REPLACE 1W1 WITH ITS X71		20W1		U142	040
		11W3		U142	050
		10X24		U142	055
		J10		U142	056
		10X71		U142	060
		J10		U142	070
		J74		U142	075
		J136		U142	076
REPLACE 1W3 WITH ITS X71		60W3		U142	080
		11W2		U142	090
ERASE HALF OF 1W2		U143		U142	100

		11W1		U142	110
		11W0		U142	120
ERASE HALF OF 1W0		U143		U142	130
		11W0		U142	140
		10X75		U142	150
TEST IF 1W1 NEEDS REVERSAL		J2		U142	160
		70	90	U142	170
		11W1		U142	180
IF SO, REVERSE		11W0		U142	190
		U148	90	U142	200
	90	11W2		U142	210
		10X76		U142	220
TEST IF 1W3 NEEDS REVERSAL		J2		U142	230
		70	91	U142	240
		11W3		U142	250
IF SO, REVERSE		11W2		U142	260
		U148	91	U142	270
COMBINE LISTS OF AFFECTED CELLS	91	11W3		U142	280
INTO NEW LIST.		11W1		U142	290
		U149		U142	300
ASSIGN COMBINED LIST		J90		U142	310
AS X71 OF NEW ROUTINE.		60W0		U142	320
		J6		U142	330
		J90		U142	332
		J136		U142	334
		60W2		U142	336
		J6		U142	338
		10X71		U142	340
		J11		U142	350
		11W2		U142	352
		10X24		U142	354
		J11		U142	356
		11W0	J33	U142	360
ERASE ATTRIBUTE NOT-(0) OF	U143	10X75		U143	000
ITEMS OF (1). NO OUTPUT		J2		U143	010
		7090		U143	020
FIND ATTRIBUTE NOT-(0).		10X76	91	U143	030
	90	10X75	91	U143	040
	91	J50		U143	050
ERASE IT.		1092		U143	060
		J100	J30	U143	070
	92	11W0		U143	080
		J14	J4	U143	090
CONSTRUCT SDSC FOR A NEW ROUTINE	U144	J50		U144	000
X75(0) TO X75(1)=OUTPUT ROUTINE		10X75		U144	010
		11W0		U144	020
		10X75		U144	030
		U142	J30	U144	040
PREFIX JDEF OF (1) TO JDEF OF (0)	U145	J51		U145	000
IN (0), AND MODIFY SDSC.		11W0		U145	010
		10X22		U145	020
FIND JDEF OF (1),		J10		U145	030
SAVE IT.		40H0		U145	040
DIVIDE IT AFTER HEAD,		J75		U145	050

	40W2		U145 060
SAVE TAIL IN W2.	20W2		U145 070
	11W1		U145 080
FIND JDEF OF (0),	10X22		U145 090
	J10		U145 100
	J74		U145 105
	J136		U145 106
INSERT IT AFTER HEAD OF JDEF (1).	J76		U145 110
	11W2		U145 120
ADD TAIL.	J76		U145 130
	30H0	J32	U145 135
CONSTRUCT SDSC FOR A NEW ROUTINE	J50		U146 000
X76(0) TO X76(1)=OUTPUT ROUTINE	10X76		U146 010
	11W0		U146 020
	10X76		U146 030
	U142	J30	U146 040
SUFFIX JDEF OF (1) TO JDEF OF	J51		U147 000
(0) IN (0), AND MODIFY SDSC	11W0		U147 010
	10X22		U147 020
FIND JDEF OF (0)	J10		U147 030
LOCATE ITS LAST SYMBOL	J61		U147 040
	11W1		U147 050
	10X22		U147 060
FIND JDEF OF (1)	J10		U147 070
	J74		U147 075
	J136		U147 076
INSERT AFTER JDEF OF (0).	J76		U147 080
	30H0	J31	U147 085
REVERSE ATTRIBUTES (0) OF ITEMS	40H0		U148 000
ON (1).	10X75		U148 010
	J2		U148 020
	70	90	U148 030
FIND ATTRIBUTE OPPOSITE TO (0).	10X75	91	U148 040
	10X76	91	U148 050
STORE (0) IN W1, OPPOSITE IN W0.	J51		U148 060
GENERATE ITEMS AND REVERSE.	1092		U148 070
	J100	J31	U148 080
	40H0		U148 090
	40H0		U148 100
	11W1		U148 110
FIND VALUE OF ATTRIBUTE,	J10		U148 120
COPY IT,	J74		U148 130
AND ASSIGN AS VALUE OF OPPOSITE	J136		U148 140
	11W0		U148 145
	J11		U148 150
ERASE ATTRIBUTE.	11W1	J14	U148 160
COMBINE LISTS OF AFFECTED CELLS.	J90		U149 000
(0)=LX75,(1)=LX76,OUTPUT=NEW LIST.	J136		U149 010
1W0=NEW LIST, 1W1=LX75,	J46		U149 020
1W2=LX76.	J22		U149 030
	11W1		U149 040
COMBINE PAIRS	1090		U149 050
	J100		U149 060
	11W1		U149 070

ADD REMAINDER OF LX75		1091		U149 080
		J100		U149 090
		11W2		U149 100
ADD REMAINDER OF LX76		1092		U149 110
		J100		U149 120
		11W0	J36	U149 130
COMBINE PAIRS	90	24W3		U149 140
		11W2		U149 150
GENERATE MEMBERS OF LX76		1093		U149 160
		J100	J4	U149 170
	93	24W4		U149 180
		11W3		U149 190
		10X41		U149 200
FIND SYMBOL OF LX75		J10		U149 210
		11W4		U149 220
		10X41		U149 230
FIND SYMBOL OF LX76		J10		U149 240
COMPARE FOR EQUALITY.		U107		U149 250
IF UNEQUAL TRY NEXT OF LX76.		70J4		U149 260
IF EQUAL,		11W1		U149 270
DELETE FROM LX75,		11W3		U149 280
		U169		U149 290
		11W2		U149 300
DELETE FROM LX76.		11W4		U149 310
		U169		U149 320
		11W4		U149 330
ADD LX75 TO LX76,		40H0		U149 340
		11W3		U149 350
		10X75		U149 360
		J10		U149 370
		10X75		U149 380
		J11		U149 390
IF PAIR IDENTICAL, DELETE		910		U149 395
		70	915	U149 396
AND INSERT PAIR IN NEW LIST.		11W0		U149 400
		J6		U149 410
STOP LX76 GENERATOR.		J64	J3	U149 420
ADD REMAINDER OF LX75	91	11W1		U149 430
		14X76		U149 435
		10X75		U149 440
		94	0	U149 450
ADD REMAINDER OF LX76	92	11W2		U149 460
		14X75		U149 465
		10X76		U149 470
		94	0	U149 480
ADD REMAINDER OF 1W3 TO 1W2	94	24W4		U149 490
		20W5		U149 500
		20W6		U149 505
		60W3		U149 510
		11W4		U149 520
		J10		U149 530
TEST FOR NORMAL FORM		96		U149 540
		7095		U149 550
IF NORMAL, DELETE	99	11W6		U149 560

	11W3		U149	570
	U169	0	U149	580
IF NOT NORMAL,	95 11W3		U149	590
CONSTRUCT PAIR ATTRIBUTE,	J90		U149	600
	J136		U149	610
	40H0		U149	620
	10X1		U149	630
INSERT IT,	J64		U149	640
	11W5		U149	650
	J11		U149	660
ADD TO 1W0	11W0		U149	670
AND DELETE FROM 1W3.	11W3		U149	680
	J64	98	U149	690
TEST IF NORMAL	96 44H0		U149	700
	J81		U149	710
	40H0		U149	720
TEST IF FIRST SYMBOL IS X1	10X1		U149	730
	J2		U149	740
IF SO, EXIT WITH H5+.	70	97	U149	750
TEST IF FIRST SYMBOL IS X2.	10X2		U149	760
	J2		U149	770
IF NOT, EXIT WITH H5-.	70J8		U149	780
IF SO, TEST IF SECOND IS X1	J82		U149	790
	10X1		U149	800
EXIT WITH H5 SET	J2	0	U149	810
	97 30H0	J8	U149	830
	98 J4	99	U149	840
	910 44H0		U149	845
	10X75		U149	850
	J10		U149	855
	40W0		U149	860
	20W0		U149	865
	40H0		U149	870
	10X76		U149	875
	J10		U149	880
	914 J60		U149	890
	70911		U149	895
	12H0		U149	900
	11W0		U149	905
	J60		U149	910
	70912		U149	915
	60W0		U149	920
	52W0		U149	925
	U107		U149	930
	70913	914	U149	935
	913 30H0	J30	U149	940
	912 30H0		U149	945
	30H0	913	U149	950
	911 30H0		U149	955
	11W0		U149	960
	J60		U149	965
	J5	913	U149	970
	915 30H0	J3	U149	975
FIND ROUTINE RELEVANT	U150 40H0		U150	000

TO DIFFERENCE (0)		10X50		U150	010
FIND TYPE OF DIFFERENCE		J10		U150	020
		10X90		U150	030
		J6		U150	040
FIND LIST OF RELEVANT ROUTINES		J10		U150	050
		J50		U150	060
		10X41		U150	065
		J10		U150	066
		10X32		U150	070
FIND TYPE OF AFFECTED CELL		J10		U150	080
		11W0		U150	090
		J6		U150	100
FIND RELEVANT ROUTINE		J10	J30	U150	110
LOCATE AFFECTED CELL (0) OF	U151	J50		U151	000
SDSC (1).		10X71		U151	010
FIND LIST OF AFFECTED CELLS		J10		U151	020
LOCATE NEXT CELL	90	J60		U151	030
IF NONE, EXIT WITH H5-		70J30		U151	040
		12H0		U151	050
FIND ITS NAME		10X41		U151	060
		J10		U151	070
		11W0		U151	080
COMPARE WITH (0).		U107		U151	090
EXIT OR CONTINUE		7090	J30	U151	100
LOCATE VARIABLE AFFECTED CELL	U152	10X71		U152	000
IN SDSC (0)		J10		U152	010
LOCATE NEXT CELL	90	J60		U152	020
IF NONE, EXIT WITH H5-		700		U152	030
		12H0		U152	040
FIND ITS NAME		10X41		U152	050
		J10		U152	060
FIND ITS TYPE		10X32		U152	070
		J10		U152	080
TEST IF VARIABLE		10X39		U152	090
		J2		U152	100
IF YES, EXIT WITH H5+		7090	J4	U152	110
SUBSTITUTE (0) FOR VARIABLE (1)	U153	J51		6W1=VAR.	U153 000
THROUGHOUT SDSC(2).		10X71		1W0=SUB.	U153 010
FIND LIST OF AFFECTED CLESS,		J10			U153 020
AND PROCESS IT.		1090			U153 030
		J100	J31		U153 040
SAVE CELL	90	40H0			U153 050
		10X41			U153 060
FIND ITS NAME		J10			U153 070
		11W1			U153 080
COMPARE IT WITH VARIABLE (1).		U107			U153 090
IF UNEQUAL, SKIP TO 91		7091			U153 100
IF EQUAL, SAVE CELL,		40H0			U153 110
ERASE ITS NAME,		10X41			U153 120
		J14			U153 130
AND SUBSTITUTE (0).		40H0			U153 140
		11W0			U153 150
		10X41			U153 160
		J11	91		U153 170

FIND INPUT OF CELL,	91	40H0		U153	180
		10X75		U153	190
		J10		U153	200
AND MAKE SUBSTITUTIONS.		92		U153	210
		10X76		U153	220
		J10		U153	230
FIND OUTPUT OF CELL,		92	J4	U153	240
AND MAKE SUBSTITUTIONS.		4J60		U153	250
SUBSTITUTE FOR CONTENTS OF LIST	92	70J8		U153	260
IF NO MORE CELLS, EXIT		12H0		U153	270
FIND FIRST SYMBOL		11W1		U153	300
AND COMPARE WITH 1W1		U107		U153	310
		7092		U153	320
IF UNEQUAL, FIND NEXT		40W2		U153	330
IF EQUAL		60W2		U153	340
		11W0		U153	350
MAKE SUBSTITUTION		J74		U153	355
		J136		U153	356
		21W2		U153	360
		30W2	92	U153	370
SUBSTITUTE NAMES OF SDSC (0) FOR	U154	J46		1W0=SDSC0	U154 000
VARIABLES OF SDSC (1) IN CELL		J22		1W1=SDSC1	U154 010
NAMED (2) OF (1).		11W0		1W2=CELL	U154 020
LOCATE CELL 1W2 IN SDSC 1W0		11W2		1W3=CELL0	U154 030
		U151		1W4=CELL1	U154 040
IF NONE, EXIT.		70J36		1W5=OUT.1	U154 050
PUT LOC. OF CELL OF W0 IN W3		20W3		U154	060
		11W1		U154	070
LOCATE CELL W2 IN SDSC W1		11W2		U154	080
		U151		U154	090
ERROR STOP		70J7		U154	100
PUT LOC OF CELL OF W1 IN W4		20W4		U154	110
		12W3		U154	120
		10X76		U154	130
FIND OUTPUT LIST OF W0 CELL		J10		U154	140
		12W4		U154	150
		10X76		U154	160
		J10		U154	170
FIND OUTPUT LIST OF W1 CELL.,		20W5		U154	180
AND STORE IN W5		90		U154	190
		12W3		U154	200
		10X75		U154	210
		J10		U154	215
		12W4		U154	220
		10X75		U154	230
		J10		U154	240
		20W5		U154	250
		90	J36	U154	260
LOCATE NEXT SYMBOL OF CELL 0	90	4J60		U154	270
IF NONE, EXIT		70J8		U154	280
		60W6		U154	285
		11W5		U154	290
LOCATE NEXT SYMBOL OF CELL 1		J60		U154	300
IF NONE, EXIT VIA 91		7091		U154	310

STORE LOCATION IN W5	60W5		U154 320
	12H0		U154 330
FIND TYPE OF SYMBOL OF 1	10X32		U154 340
	J10		U154 350
TEST IF A VARIABLE	10X39		U154 360
	J2		U154 370
IF NOT, GO TO 90 VIA 93	7092		U154 380
	52H0		U154 390
BRING IN SDSC 1	11W1		U154 420
	J6		U154 430
	12W6		U154 440
SUBSTITUTE THROUGH SDSC 1	U153	90	U154 470
	91 30H0	J8	U154 480
	92 30H0	90	U154 490
MAKE JDEF OF (0). (0) IS ROUTINE	U155 J47		U155 000
	J21		U155 005
WITH SDSC MODIFIED FROM (1).	11W0		U155 010
	11W1		U155 020
FIND JDEF OF (1)	10X22		U155 030
	J10		U155 040
	J74		U155 045
	J136		U155 046
ASSIGN IT AS JDEF OF (0)	10X22		U155 050
	J11		U155 060
BRING IN LISTS OF AFFECTED CELLS	93	91	U155 065
	93 04J0		U155 066
	11W0		U155 070
	10X24		U155 080
FIND SDSC OF (0)	J10		U155 090
	10X71		U155 100
FIND LIST OF AFFECTED CELLS	J10		U155 110
AND PUT IN W2	20W2		U155 120
	11W1		U155 130
	10X24		U155 140
FIND SDSC OF (1)	J10		U155 150
	10X71		U155 160
FIND LIST OF AFFECTED CELLS	J10		U155 170
AND PUT IN W3	60W3		U155 180
	11W2	0	U155 190
FIND NEXT CELL OF (0)	91 J60		U155 200
	7090		U155 210
	11W3		U155 220
FIND NEXT CELL OF (1)	J60		U155 230
	60W3		U155 240
	70J7		U155 250
	52H0		U155 260
	40H0		U155 220
FIND ITS NAME	10X41		U155 280
	J10		U155 290
TEST IF IT IS A VARIABLE	10X32		U155 300
	J10		U155 310
	10X39		U155 320
	J2		U155 330
	7092		U155 340

SO, STORE IN W4		20W4		U155 350
STORE NAME OF SYMBOL OF (0) IN W5		12H0		U155 360
		20W5		U155 370
		99	91	U155 375
SUBSTITUTE W5 FOR W4	99	04J0		U155 380
		11W0		U155 385
		10X22		U155 385
		J10		U155 386
THROUGHOUT (0)		11W4		U155 390
		10X41		U155 395
		J10		U155 396
		11W5		U155 400
		10X41		U155 405
		J10		U155 406
		U156	0	U155 410
	92	30H0	91	U155 420
	90	30H0		U155 430
		30H0		U155 440
		93		U155 450
SUBSTITUTE FOR VARIABLES IN CELLS	94	J60		U155 460
		7095		U155 470
		60W2		U155 480
		52H0		U155 490
		10X75		U155 500
		J10		U155 510
PUT X75 OF W2 IN W6		20W6		U155 520
		J60		U155 530
		70J7		U155 540
		60W3		U155 550
		52H0		U155 560
		10X75		U155 570
		J10		U155 580
PUT X75 OF W3 IN W7		20W7		U155 590
SUBS. FOR VARIABLE IN X75		96		U155 600
		12W2		U155 610
		10X76		U155 620
		J10		U155 630
PUT X76 OF W2 IN W6		20W6		U155 640
		12W3		U155 650
		10X76		U155 660
		J10		U155 670
PUT X76 OF W3 IN W7		20W7		U155 680
SUBS. FOR VARIABLE IN X76		96		U155 690
		11W3		U155 700
		11W2	94	U155 710
EXIT	95	30H0		U155 720
		30H0	J37	U155 730
FIND VARIABLES AND SUBSTITUTE	96	04J0		U155 740
		11W6		U155 745
FIND NEXT IN W6		J60		U155 750
IF NONE, EXIT		70J8		U155 760
		60W6		U155 770
		11W7		U155 780
		J60		U155 790

	70J7		U155 800
	60W7		U155 810
TEST IF NEXT IN W7 IS VARIABLE	12H0		U155 820
	10X32		U155 830
	J10		U155 840
	10X39		U155 850
	J2		U155 860
IF NOT, FIND NEXT	7098		U155 870
IF SO,	52H0		U155 880
SUBSTITUTE W5 FOR W4	20W4		U155 890
THROUGHOUT W0	52H0		U155 900
	20W5		U155 910
	11W0		U155 920
	10X22		U155 921
	J10		U155 922
	11W4		U155 924
	11W5		U155 926
	U156	96	U155 928
	98 30H0		U155 930
	30H0	96	U155 940
SUBSTITUTE SYMBOL (0) FOR	U156 J52		U156 000
VARIABLE (1) IN JDEF OF (2)	11W2		U156 010
FIND NEXT LINE OF JDEF	90 J60		U156 020
IF NONE, EXIT	7091		U156 030
	12H0		U156 040
FIND ITS SYMB.	10X45		U156 050
	J10		U156 060
COMPARE SYMB WITH 1W1	11W1		U156 070
	U107		U156 080
IF UNEQUAL, FIND NEXT	7090		U156 090
IF EQUAL,	12H0		U156 100
COPY 1W0 AS NEW SYMB	11W0		U156 110
FOR LINE OF JDEF	J74		U156 120
	J136		U156 130
	10X45		U156 140
	J11	90	U156 210
	91 30H0	J32	U156 220
DELETE CAREFULLY SYMBOL (0)	U169 J41		U169 000
FROM LIST (1)	20W0		U169 010
	40H0		U169 020
	11W0		U169 030
LOCATE (0) ON (1)	J62		U169 040
	70J30		U169 050
DIVIDE LIST AFTER LOC(0)	J75		U169 060
PUT REMAINDER IN W1	20W1		U169 070
DELETE LAST SYMBOL FROM (1)	40H0		U169 080
	J70		U169 090
LOCATE LAST SYMBOL OF (1)	J61		U169 100
	11W1		U169 110
REATTACH REMAINDER	J76		U169 120
	30H0	J31	U169 130
FIND BIGGEST DIFFERENCE	x5 J43		X5 000
BETWEEN DSNS (0) AND (1)	20W0		X5 010
	60W1		X5 020

		10X30		X5	030
FIND PROCESS OF (1)		J10		X5	040
		11W0		X5	050
		10X30		X5	060
FIND PROCESS OF (0)		J10		X5	070
COMPARE PROCESSES		J2		X5	080
IF EQUAL, COMPARE ARGUMENTS		70	90	X5	090
IF UNEQUAL, SIGNAL X62		10X62	J33	X5	100
FIND ARG. LISTS OF (0) AND (1)	90	910		X5	110
FIND NEXT ARGS. OF (0) AND (1)	92	911		X5	120
NONE, GO TO 91		7091		X5	130
TEST ARGUMENT TYPES		X6		X5	140
IF UNEQUAL, EXIT WITH SIGNAL		70J33	92	X5	150
ALL TYPES EQUAL, TEST ARGS.	91	910		X5	160
FIND NEXT ARGS. OF (0) AND (1)	93	911		X5	170
NONE, GO TO 94		7094		X5	180
		40H0		X5	190
		10X32		X5	200
FIND ARGUMENT TYPE		J10		X5	210
		70J7		X5	220
		10X37		X5	230
TEST IF DETERMINER		J2		X5	240
IF NOT, FIND NEXT ARG.		7093		X5	250
IF SO, FIND BIGGEST DIFFERENCE		X5		X5	260
SAVE DIFFERENCE SIGNAL		40H0		X5	270
TEST IF NO DIFFERENCE		10X55		X5	280
		J2		X5	290
IF DIFFERENCE, EXIT WITH SIGNAL		7097		X5	300
NO DIFFERENCE, FIND NEXT ARG.		30H0	93	X5	310
NO DETERMINER DIFFERENCES	94	910		X5	320
FIND NEXT ARGS. OF (0) AND (1)		911		X5	330
NONE, EXIT WITH X55		7095		X5	340
		40H0		X5	350
TEST IF NAMES		10X36		X5	360
		J2		X5	370
		7096		X5	380
SIGNAL DIFFERENT NAMES		10X66	J33	X5	390
SIGNAL DIFFERENT LOCATIONS	96	10X67	J33	X5	400
EXIT, NO DIFFERENCE	95	10X55	J33	X5	410
FIND ARGUMENT LISTS	910	11W0		X5	420
		10X31		X5	430
		J10		X5	440
FIND LIST OF (0)		20W2		X5	450
AND PUT IN W2.		11W1		X5	460
		10X31		X5	470
		J10		X5	480
FIND LIST OF (1),		20W3	0	X5	490
AND PUT IN W3.		11W2		X5	500
FIND NEXT ARGS. OF (0) AND (1)	911	J60		X5	510
FIND NEXT OF (0)		700		X5	520
IF NONE, SET H5 -		60W2		X5	530
		52H0		X5	540
		11W3		X5	550
FIND NEXT OF (1)		J60		X5	560

		70J7		X5	570
		60W3		X5	580
		52H0	0	X5	590
TEST TYPES OF ARGUMENTS (0), (1)	X6	J43		X6	000
OUTPUT (0) IS DIFFERENCE, SET H5.		20W0		X6	010
		60W1		X6	020
		10X32		X6	030
FIND ARGUMENT TYPE OF (1)		J10		X6	040
STORE IN W3		60W3		X6	045
		11W0		X6	050
		10X32		X6	060
FIND ARGUMENT TYPE OF (0)		J10		X6	070
STORE IN W2		60W2		X6	075
COMPARE ARGUMENT TYPES		J2		X6	080
		70	J33	X6	090
UNEQUAL, TEST IF (0) DETERMINER		11W2		X6	100
		10X37		X6	110
		J2		X6	120
		7091		X6	130
SIGNAL (0) IS DETERMINER		10X65	90	X6	140
TEST IF (0) IS A NAME	91	11W2		X6	150
		10X36		X6	160
		J2		X6	170
		7092		X6	180
SIGNAL (0) A NAME		10X68	90	X6	190
IF (0) A LOCATION, SIGNAL X69	92	10X69	J33	X6	200
	90	J5	J33	X6	210
FIND LARGEST DIFFERENCE	X7	J44		X7	000
BETWEEN INPUT AND OUTPUT		10X71		X7	010
OF SDSC (0). PUT DIFF. IN (0).		J10		X7	020
PUT LIST OF AFFECTED CELLS IN W0.		60W0		X7	030
FIND DIFFERENCE, NOT IN H0.		1090		X7	040
		J100		X7	050
EXIT IF FOUND		70J34		X7	060
		11W0		X7	070
FIND DIFFERENCE IN H0		1091		X7	080
		J100		X7	090
EXIT IF FOUND		70J34		X7	100
		J90		X7	110
IF NOT FOUND,		J136		X7	120
		40H0		X7	125
LEAVE NULL DIFFERENCE,		10X80		X7	130
V(X50)=X80, IN H0		10X50		X7	140
		J11	J34	X7	150
PUT AFFECTED CELL IN W1	90	64W1		X7	160
		10X41		X7	170
FIND TYPE OF AFFECTED CELL		J10		X7	180
COPY AND SAVE ITS NAME		J74		X7	182
		J136		X7	183
		40H0		X7	184
		10X32		X7	190
		J10		X7	200
TEST IF H0		10X35		X7	220
		J2		X7	230

IF SO, EXIT.		7092	J8	X7	240
IF NOT, CREATE DIFFERENC	92	J90		X7	250
		J136		X7	260
SAVE ITS NAME IN W2,		60W2		X7	270
		J6		X7	280
ASSIGN TYPE OF CELL		10X41		X7	290
		J11		X7	300
		11W1		X7	310
FIND TYPE OF DIFFERENCE		93		X7	320
		11W2		X7	330
		J6		X7	340
		10X50		X7	350
		J11		X7	360
ASSIGN TYPE TO 1W2		11W2	J3	X7	370
BRING DIFFERENCE IN, AND QUIT		64W1		X7	380
FIND TYPE OF AFFECTED CELL	91	10X41		X7	390
		J10		X7	400
		J74		X7	410
COPY ITS NAME		J136		X7	420
		J3	92	X7	430
SAVE AFFECTED CELL	93	44H0		X7	440
		10X75		X7	450
		J10		X7	460
FIND ITS FIRST INPUT SYMBOL,		J81		X7	470
AND SAVE IN W3.		20W3		X7	480
SAVE AFFECTED CELL		40H0		X7	490
		10X76		X7	500
		J10		X7	510
FIND ITS FIRST OUTPUT SYMBOL		J81		X7	520
		11W3		X7	530
COMPARE SYMBOLS		U107		X7	540
		7094		X7	550
IF IDENTICAL,		30H0		X7	560
CALL DIFFERENCE (COPY)		10X84	0	X7	570
NOT IDENTICAL,	94	40H0		X7	580
COUNT LIST X75		10X75		X7	590
		J10		X7	600
		J126		X7	610
		20W3		X7	620
COUNT LIST X76		10X76		X7	630
		J10		X7	640
		J126		X7	650
		60W4		X7	660
		11W3		X7	670
		J114		X7	680
IF LISTS EQUAL		7095		X7	690
CALL DIFFERENCE (REPLACE)		10X83	0	X7	700
	95	11W3		X7	710
		11W4		X7	720
		J115		X7	730
IF 1W4 LARGER,		7096		X7	740
CALL DIFFERENCE (ADD)		10X82	0	X7	750
IF SMALLER, (DELETE)	96	10X81	0	X7	760
DATA	5	1			

A	A0	0	0	A0	000
B	L1	21A		L1	000
C	L2	21B		L2	000
D	L3	21C		L3	000
E	L4	21D		L4	000
F	L5	21E		L5	000
G	L6	21F		L6	000
H	L7	21G		L7	000
I	L8	21H		L8	000
J	L9	21I		L9	000
K	L10	21J		L10	000
L	L11	21K		L11	000
M	L12	21L		L12	000
N	L13	21M		L13	000
O	L14	21N		L14	000
P	L15	21O		L15	000
Q	L16	21P		L16	000
R	L17	21Q		L17	000
S	L18	21R		L18	000
T	L19	21S		L19	000
U	L20	21T		L20	000
V	L21	21U		L21	000
W	L22	21V		L22	000
X	L23	21W		L23	000
Y	L24	21X		L24	000
Z	L25	21Y		L25	000
0	L26	21Z		L26	000
1	L30	210		L30	000
2	L31	211		L31	000
3	L32	212		L32	000
4	L33	213		L33	000
5	L34	214		L34	000
6	L35	215		L35	000
7	L36	216		L36	000
8	L37	217		L37	000
9	L38	218		L38	000
(0)	L39	219		L39	000
(1)	L40	21(0)		L40	000
(2)	L41	21(1)		L41	000
(3)	L42	21(2)		L42	000
(4)	L43	21(3)		L43	000
	L44	21(4)		L44	000
	N0	1	0000	N0	000
	N1	1	0001	N1	000
	N2	1	0002	N2	000
	N3	1	0003	N3	000
	N4	1	0004	N4	000
	N5	1	0005	N5	000
	N6	1	0006	N6	000
	N7	1	0007	N7	000
	N8	1	0008	N8	000
	N9	1	0009	N9	000
	N10	1	0010	N10	000
	N13	1	0013	N13	000

	N28	1	0028	N28	000
	N29	1	0029	N29	000
	N30	1	0030	N30	000
	N35	1	0035	N35	000
	N36	1	0036	N36	000
	N37	1	0037	N37	000
	N38	1	0038	N38	000
	N44	1	0044	N43	000
	N49	1	0049	N49	000
	N50	1	0050	N50	000
	N60	1	0060	N60	000
	N65	1	0065	N65	000
	N77	1	0077	N77	000
	N116	1	0116	N116	000
DIFFERENCE	X0	0	0	X0	000
REMAINDER OF PUSHDOWN LIST	X1	90	0	X1	000
	90	0		X1	010
		X32		X1	020
		X1		X1	030
		X33		X1	040
		L18		X1	050
		X34		X1	060
		N0	0	X1	070
VARIABLE S1	X2	90	0	X2	000
	90	0		X2	010
		X32		X2	020
		X38		X2	030
		X33		X2	040
		L19		X2	050
		X34		X2	060
		N1	0	X2	070
VARIABLE S0	X3	90	0	X3	000
	90	0		X3	010
		X32		X3	020
		X39		X3	030
		X33		X3	040
		L19		X3	050
		X34		X3	060
		N0	0	X3	070
VARIABLE S2	X4	90	0	X4	000
	90	0		X4	010
		X32		X4	020
		X39		X4	030
		X33		X4	040
		L19		X4	050
		X34		X4	060
		N2	0	X4	070
DSCN (DESCRIPTIVE NAME)	X20	0	0	X20	000
LDEF (L-DEFINITION)	X21	0	0	X21	000
JDEF (J-DEFINITION)	X22	0	0	X22	000
ASQJ (ASSOCIATED J)	X23	0	0	X23	000
SDSC OF ROUTINE	X24	0	0	X24	000
IPLN OF ROUTINE	X25	0	0	X25	000
PROCESS	X30	0	0	X30	000

LIST OF ARGUMENTS	X31	0	0	X31	000
TYPE (OF ARGUMENT)	X32	0	0	X32	000
REGION	X33	0	0	X33	000
LOCATION	X34	0	0	X34	000
MEMBER OF HO LIST	X35	0	0	X35	000
NAMED SYMBOL	X36	0	0	X36	000
DETERMINER	X37	0	0	X37	000
UNKNOWN CONSTANT	X38	0	0	X38	000
VARIABLE	X39	0	0	X39	000
TYPE (OF IPL WORD)	X40	0	0	X40	000
NAME	X41	0	0	X41	000
SIGN	X42	0	0	X42	000
P-PREFIX	X43	0	0	X43	000
Q-PREFIX	X44	0	0	X44	000
SYMB	X45	0	0	X45	000
LINK	X46	0	0	X46	000
TYPE OF DIFFERENCE	X50	0	0	X50	000
TEST FOR DIFFERENCE	X51	0	0	X51	000
NO DIFFERENCE	X55	0	0	X55	000
JDEF ABSENT	X60	0	0	X60	000
LDEF ABSENT	X61	0	0	X61	000
DSCN-WRONG FUNCTION NAME	X62	0	0	X62	000
DSCN-WRONG ARGUMENT TYPE	X63	0	0	X63	000
LDEF HAS CONNECTIVE	X64	0	0	X64	000
DSCN-DETERM. FOR OTHER	X65	0	0	X65	000
DSCN-ONE NAME FOR OTHER	X66	0	0	X66	000
DSCN-ONE LOC FOR OTHER	X67	0	0	X67	000
DSCN-NAME FOR LOCATION	X68	0	0	X68	000
DSCN-LOCATION FOR NAME	X69	0	0	X69	000
OPERATOR ROUTINE	X70	0	0	X70	000
LIST OF AFFECTED CELLS OF SDSC	X71	0	0	X71	000
INPUT STATE OF CELL	X75	0	0	X75	000
OUTPUT STATE OF CELL	X76	0	0	X76	000
SDSC - NO DIFFERENCE	X80	0	0	X80	000
SDSC - DELETION	X81	0	0	X81	000
SDSC - ADDITION	X82	0	0	X82	000
SDSC - REPLACEMENT	X83	0	0	X83	000
SDSC - COPY	X84	0	0	X84	000
INDEX TO RELEVANT ROUTINES	X90	90	0	X90	000
	90	0		X90	050
LIST OF ADD ROUTINES		X82		X90	100
		91		X90	110
LIST OF REPLACE ROUTINES		X83		X90	200
		92	0	X90	210
	91	93	0	X90	300
ADD.	93	0		X90	350
HO IS AFFECTED CELL		X35		X90	400
		X107	0	X90	410
	92	94	0	X90	500
REPLACE.	94	0		X90	550
NAMED CELL IS AFFECTED CELL		X36		X90	600
		X106	0	X90	610
ROUT. INSERT (1) AT END OF	X100	90	0	X100	000
VALUE LIST OF ATTRIBUTE	90	0		X100	010

		X25		X100 015
		97		X100 016
		X20		X100 020
(0) OF (2).		91	0	X100 030
DSCN IS 91.	91	92	0	X100 040
	92	0		X100 050
PROCESS IS X110		X30		X100 060
		X110		X100 070
		X31		X100 080
		93	0	X100 090
ARGS. ARE	93	0		X100 100
X120		X120		X100 110
X121		X121	0	X100 120
	97	99	0	X100 210
	99	0		X100 220
SYMB. = J13		X33		X100 230
		L10		X100 240
		X34		X100 250
		N13	0	X100 260
ROUT. INSERT (0) AT END OF (1)	X101	90	0	X101 000
	90	0		X101 010
		X20		X101 020
		91		X101 030
JDEF IS 94		X22		X101 035
		94	0	X101 036
	91	92	0	X101 040
	92	0		X101 050
PROCESS IS X110		X30		X101 060
		X110		X101 070
		X31		X101 080
		93	0	X101 090
	93	0		X101 100
		X122		X101 110
		X120	0	X101 120
	94	0		X101 130
		95	0	X101 140
	95	96	0	X101 150
	96	0		X101 160
J65		X45		X101 170
		97		X101 180
		X46		X101 190
		98	0	X101 200
	97	99	0	X101 210
PROCESS J65	99	0		X101 220
		X33		X101 230
		L10		X101 240
		X34		X101 250
		N65	0	X101 260
	98	910	0	X101 270
LINK 0	910	0		X101 280
		X33		X101 290
		N0	0	X101 300
ROUT. FIND V((0),(1))	X102	90	0	X102 000
	90	0		X102 010

TYPE--DETERMINER		X32		X102 020
		X37		X102 030
		X20		X102 040
JDEF IS 94		91		X102 050
		X22		X102 060
		94	0	X102 070
VERB	91	92		X102 080
		920	0	X102 085
	92	0		X102 090
		X30		X102 100
PROCESS IS X111. FIND VALUE		X111		X102 110
		X31		X102 120
		93	0	X102 130
ARGS. ARE	93	0		X102 140
X122. (0)		X122		X102 150
X120. (1)		X120	0	X102 160
	94	0		X102 170
		95	0	X102 180
	95	96	0	X102 190
	96	0		X102 200
		X45		X102 210
		97		X102 220
		X46		X102 230
		98	0	X102 240
	97	99	0	X102 250
PROCESS J10	99	0		X102 260
		X33		X102 270
		L10		X102 280
		X34		X102 290
		N10	0	X102 300
	98	910	0	X102 310
LINK 0	910	0		X102 320
		X33		X102 330
		N0	0	X102 340
VERB (FIND)	920	21FIND		X102 350
ROUT. TESTN IF (0) LESS THAN NO	X103	90	0	X103 000
	90	0		X103 010
		X20		X103 020
		91		X103 030
		X22		X103 040
		94	0	X103 050
	91	92	0	X103 060
	92	0		X103 070
		X30		X103 080
PROCESS IS X112 TESTN, LESS		X112		X103 090
		X31		X103 100
		93	0	X103 110
ARGUMENTS	93	0		X103 120
(0)		X122		X103 130
NO		X124	0	X103 140
	94	0	0	X103 150
ROUT. TESTN IF (0) LESS THAN (1)	X104	90	0	X104 000
	90	0		X104 010
		X20		X104 020

	91		X104 030
	X22		X104 040
	94	0	X104 050
91	92	0	X104 060
92	0		X104 070
	X30		X104 080
PROCESS IS X112 TESTN, LESS	X112		X104 090
	X31		X104 100
	93	0	X104 110
	0		X104 120
93	X122		X104 130
	X120	0	X104 140
94	0		X104 150
	95	0	X104 160
95	96	0	X104 170
96	0		X104 180
	X45		X104 190
	97		X104 200
	X46		X104 210
	98	0	X104 220
97	99	0	X104 230
99	0		X104 240
	X33		X104 250
	L10		X104 260
	X34		X104 270
	N116	0	X104 280
98	910	0	X104 290
910	0		X104 300
	X33		X104 310
	N0	0	X104 320
J3. REPLACE TOP OF H5 WITH J3.	X105	90	X105 000
90	0		X105 010
	X20		X105 015
	920		X105 016
	X25		X105 020
	91		X105 030
	X24		X105 040
	92	0	X105 050
IPLN OF J3	93	0	X105 060
	0		X105 070
	X33		X105 080
	L10		X105 090
	X34		X105 100
	N3	0	X105 110
SDSC OF J3	92	94	X105 120
	94	0	X105 130
	X71		X105 140
	95	0	X105 150
	0		X105 160
AFFECTED CELL IS 97	97	0	X105 170
	97	98	X105 180
	98	0	X105 190
	X41		X105 200
	X175		X105 210

		X75		X105 220
		99		X105 230
		X76		X105 240
		910	0	X105 250
INPUT STATE OF H5	99	0		X105 260
		X2		X105 270
		X1	0	X105 280
OUTPUT STATE OF H5.	910	0		X105 290
		911		X105 300
		X1	0	X105 310
	911	912	0	X105 320
	912	0		X105 330
		X32		X105 335
		X36		X105 336
		X33		X105 340
		L10		X105 350
		X34		X105 360
		N3	0	X105 370
	920	921	0	X105 380
DSCN	921	0		X105 390
		X30		X105 400
PROCESS IS X113		X113	0	X105 410
P2(C). REPLACE TOP OF C WITH	X106	90	0	X106 000
TOP OF H0.	90	0		X106 010
DSCN		X20		X106 012
		930		X106 013
		X22		X106 015
		920		X106 016
		X25		X106 020
		91		X106 030
		X24		X106 040
		92	0	X106 050
IPLN OF P2(C)	91	93	0	X106 060
	93	0		X106 070
		X33		X106 080
		L16		X106 090
		X34		X106 100
		N2	0	X106 110
SDSC OF P2(C)	92	94	0	X106 120
	94	0		X106 130
		X71		X106 140
		95	0	X106 150
	95	0		X106 160
AFFECTED CELLS ARE		96		X106 170
96 AND 97		97	0	X106 180
	96	98	0	X106 190
	98	0		X106 200
		X41		X106 210
		X170		X106 220
		X75		X106 240
		99		X106 250
		X76		X106 260
		910	0	X106 270
INPUT STATE OF H0	99	0		X106 280

		X3		X106 290
		X1	0	X106 300
OUTPUT STATE OF H0	910	0		X106 310
		X1	0	X106 320
	97	914	0	X106 330
	914	0		X106 340
		X41		X106 341
NAME IS VARIABLE, X4		X4		X106 342
		X75		X106 350
		915		X106 360
		X76		X106 370
		916	0	X106 380
INPUT STATE OF C	915	0		X106 390
		X2		X106 400
		X1	0	X106 410
OUTPUT STATE OF C	916	0		X106 420
		X3		X106 430
		X1	0	X106 440
	920	0		X106 450
		921	0	X106 460
	921	922	0	X106 470
	922	0		X106 480
		X43		X106 490
		N2		X106 500
		X45		X106 510
		X4	0	X106 520
	930	0		X106 530
		X30		X106 540
PROCESS IS X114.		X114		X106 550
		X31		X106 560
		931	0	X106 570
	931	0		X106 580
ARG. IS X125.		X125	0	X106 590
P1(S). ADD S TO TOP OF H0.	X107	90	0	X107 000
	90	0		X107 010
		X20		X107 012
		930		X107 013
		X22		X107 015
		920		X107 016
		X25		X107 020
		91		X107 030
		X24		X107 040
		92	0	X107 050
IPLN OF P1(S)	91	93	0	X107 060
	93	0		X107 070
		X33		X107 080
		L16		X107 090
		X34		X107 100
		N1	0	X107 110
SDSC OF P1(S)	92	94	0	X107 120
	94	0		X107 130
		X71		X107 140
		95	0	X107 150
	95	0		X107 160

	97	0	X107 170
AFFECTED CELL IS H0	97	98 0	X107 180
	98	0	X107 190
		X41	X107 200
		X170	X107 210
		X75	X107 220
		99	X107 230
		X76	X107 240
		910 0	X107 250
INPUT STATE OF H0	99	0	X107 260
		X1 0	X107 270
OUTPUT STATE OF H0	910	0	X107 280
		X3	X107 290
		X1 0	X107 300
	920	0	X107 310
		921 0	X107 320
	921	922 0	X107 330
	922	0	X107 340
		X43	X107 350
		N1	X107 360
		X45	X107 370
		X3 0	X107 380
DSCN	930	0	X107 390
		X30	X107 400
PROCESS IS X115.		X115	X107 410
		X31	X107 420
		931 0	X107 430
	931	0	X107 440
ARG. IS X126		X126 0	X107 450
	X110	0	X110 000
		911	X110 010
		912	X110 020
		X31	X110 030
		914	X110 040
		915	X110 050
		X31 0	X110 060
	911	211NSER	X110 001
	912	21T	X110 002
	914	21AT EN	X110 004
	915	21D OF	X110 005
	X111	0	X111 000
		911	X111 010
		912	X111 020
		913	X111 030
		914	X111 040
		915	X111 050
		916	X111 060
		X31	X111 070
		918	X111 080
		X31 0	X111 090
	911	21THE V	X111 001
	912	21ALUE	X111 002
	913	21-OF T	X111 003
	914	21HE AT	X111 004

	915	21TRIBU		X111	005
	916	21TE		X111	006
	918	21OF		X111	008
PROCESS TESTN () LESS ()	X112	0		X112	000
		911		X112	010
		912		X112	020
		X31		X112	030
		914		X112	040
		915		X112	050
		916		X112	060
		X31	0	X112	070
	911	21TEST-		X112	080
	912	21N IF		X112	090
	914	21IS LE		X112	100
	915	21SS TH		X112	110
	916	21AN		X112	120
PROCESS. SET H5 -	X113	0		X113	000
		911		X113	010
		912		X113	020
		913	0	X113	030
	911	21SET H		X113	040
	912	215 MIN		X113	050
	913	21US		X113	060
PROCESS. OUTPUT (0) TO ().	X114	0		X114	000
		911		X114	010
		912		X114	020
		913		X114	030
		X31	0	X114	040
	911	21OUTPU		X114	050
	912	21T (0)		X114	060
	913	21 TO		X114	070
PROCESS. INPUT ().	X115	0		X115	000
		911		X115	010
		X31	0	X115	020
	911	21INPUT		X115	030
ARG. X120 (1)	X120	90		X120	000
		92	0	X120	005
	90	0		X120	010
		X32		X120	020
TYPE- H0 LIST LOCATION		X35		X120	030
		X37		X120	040
		91		X120	050
		X45		X120	060
		X170	0	X120	070
	91	0		X120	080
		X151		X120	090
		X150	0	X120	100
	92	21(1)		X120	110
ARG. X121	X121	90	0	X121	000
	90	0		X121	010
TYPE- DETERMINER		X32		X121	020
		X37		X121	030
		X22		X121	035
JDEF IS 94		94		X121	036

DSCN IS 91.

	X20		X121 040
	91	0	X121 050
91	92	0	X121 060
92	0		X121 070

PROCESS IS X111.

	X30		X121 080
	X111		X121 090
	X31		X121 100
	93	0	X121 110
93	0		X121 120
	X122		X121 130
	X123	0	X121 140

ARGS. ARE

X122

X123

94	0		X121 150
	95	0	X121 160
95	96	0	X121 170
96	0		X121 180
	X45		X121 190
	97		X121 200
	X46		X121 210
	98	0	X121 220
97	99	0	X121 230
99	0		X121 240
	X33		X121 250
	L10		X121 260
	X34		X121 270
	N10	0	X121 280
98	910	0	X121 290
910	0		X121 300

ARG. X122 (0)

	X33		X121 310
	N0	0	X121 320
X122	90		X122 000
	92	0	X122 005
90	0		X122 010

TYPE- HQ LIST LOCATION

	X32		X122 020
	X35		X122 030
	X37		X122 040
	91		X122 050
	X45		X122 060
	X170	0	X122 070
91	0		X122 080
	X151	0	X122 090

ARG. X123 (2)

92	21(0)		X122 100
X123	90		X123 000
	92	0	X123 005
90	0		X123 010

TYPE- HQ LIST LOCATION

	X32		X123 020
	X35		X123 030
	X37		X123 040
	91		X123 050
	X45		X123 060
	X170	0	X123 070
91	0		X123 080
	X151		X123 090
	X150		X123 100
	X150	0	X123 110

	92	21(2)		X123	120
ARG. X124 ONO	X124	90	0	X124	000
	90	0		X124	010
		X32		X124	020
		X36		X124	030
		X37		X124	040
		91		X124	050
		X45		X124	060
		92	0	X124	070
	91	0	0	X124	080
	92	93	0	X124	090
	93	0		X124	100
		X33		X124	110
		L14		X124	120
		X34		X124	130
		N0	0	X124	140
ARG. X125. VARIABLE S2.	X125	90	0	X125	000
	90	0		X125	010
		X32		X125	020
		X36		X125	030
		X37		X125	040
		91		X125	050
		X45		X125	060
		X4	0	X125	070
	91	0	0	X125	080
ARG. X126. VARIABLE S0.	X126	90	0	X126	000
	90	0		X126	010
		X32		X126	020
		X36		X126	030
		X37		X126	040
		91		X126	050
		X45		X126	060
		X3	0	X126	070
	91	0	0	X126	080
	X150	0		X150	000
		911		X150	010
		912		X150	020
		913	0	X150	030
	911	21THE N		X150	001
	912	21EXT O		X150	002
	913	21F		X150	
	X151	0		X151	000
		911		X151	010
		912		X151	020
		913	0	X151	030
	911	21THE C		X151	001
	912	21ONTEN		X151	002
	913	21TS OF		X151	003
	X170	90	0	X170	000
SYMB H0	90	0		X170	010
		X32		X170	015
		X35		X170	016
		X33		X170	020
		L8		X170	030

		X34		X170	040
		N0	0	X170	050
SYMB H1	X171	90	0	X171	000
	90	0		X171	010
		X32		X171	015
		X36		X171	016
		X33		X171	020
		L8		X171	030
		X34		X171	040
		N1	0	X171	050
SYMB H2	X172	90	0	X172	000
	90	0		X172	010
		X32		X172	015
		X36		X172	016
		X33		X172	020
		L8		X172	030
		X34		X172	040
		N2	0	X172	050
SYMB. H5	X175	90	0	X175	000
	90	0		X175	010
		X32		X175	015
		X36		X175	016
		X33		X175	020
		L8		X175	030
		X34		X175	040
		N5	0	X175	050
LIST OF SEGMENTS FOR	X180	0		X180	000
COMPOSING J77		90		X180	005
		91		X180	010
		92		X180	015
		93	0	X180	020
	90	0		X180	025
		94	0	X180	030
	94	940	0	X180	035
	940	0		X180	040
		X45		X180	045
		941	0	X180	050
	941	942	0	X180	055
	942	0		X180	060
		X33		X180	065
		L10		X180	070
		X34		X180	075
		N50	0	X180	080
	91	0		X180	085
		95	0	X180	090
	95	950	0	X180	095
	950	0		X180	100
		X45		X180	105
		951	0	X180	110
	951	952	0	X180	115
	952	0		X180	120
		X33		X180	125
		L10		X180	130
		X34		X180	135

	N60	0	X180	140
92	0		X180	145
	96		X180	150
	97		X180	155
	98	0	X180	160
96	960	0	X180	165
960	0		X180	170
	X43		X180	175
	N1		X180	180
	X44		X180	185
	N2		X180	190
	X45		X180	195
	961	0	X180	200
	962	0	X180	205
962	0		X180	210
	X33		X180	215
	L8		X180	220
	X34		X180	225
	N0	0	X180	230
97	970	0	X180	235
970	0		X180	240
	X43		X180	245
	N1		X180	250
	X44		X180	255
	N1		X180	260
	X45		X180	265
	971	0	X180	270
	972	0	X180	275
972	0		X180	280
	X33		X180	285
	L23		X180	290
	X34		X180	295
	N0	0	X180	300
98	980	0	X180	305
980	0		X180	310
	X45		X180	315
	981	0	X180	320
981	982	0	X180	325
982	0		X180	330
	X33		X180	335
	L10		X180	340
	X34		X180	345
	N2	0	X180	350
93	0		X180	355
	99		X180	360
	910	0	X180	365
99	990	0	X180	370
990	0		X180	375
	X43		X180	380
	N3		X180	385
	X44		X180	390
	N0		X180	395
	X45		X180	400
	991	0	X180	405

FLOW DIAGRAM FOR J77

991	992	0	X180	410
992	0		X180	415
	X33		X180	420
	L8		X180	425
	X34		X180	430
	N0	0	X180	435
910	995	0	X180	440
995	0		X180	445
	X45		X180	450
	996	0	X180	455
996	997	0	X180	460
997	0		X180	465
	X33		X180	470
	L10		X180	475
	X34		X180	480
	N30	0	X180	485
X181	0		X181	000
	90		X181	010
	91		X181	020
	92		X181	030
	93	0	X181	040
90	905	0	X181	050
905	0		X181	060
	X41		X181	070
	N1		X181	080
	X43		X181	090
	N0		X181	100
	X46		X181	110
	N2	0	X181	120
91	915	0	X181	130
915	0		X181	140
	X41		X181	150
	N2		X181	160
	X43		X181	170
	N7		X181	180
	X45		X181	190
	N4		X181	200
	X46		X181	210
	N3	0	X181	220
92	925	0	X181	230
925	0		X181	240
	X41		X181	250
	N3		X181	260
	X43		X181	270
	N7		X181	280
	X45		X181	290
	N2		X181	300
	X46		X181	310
	N4	0	X181	320
93	935	0	X181	330
935	0		X181	340
	X41		X181	350
	N4		X181	360
	X43		X181	370

		NO		X181 380
		X46		X181 390
		NO	0	X181 400
ROUTINE J77	X182	90	0	X182 000
	90	0		X182 010
		X41		X182 020
		91	0	X182 030
	91	92	0	X182 040
	92	0		X182 050
		X33		X182 055
		L10		X182 060
		X34		X182 070
		N77	0	X182 080
LIST OF COMPILED ROUTINES	X196	0		X196 000
WITH DSCNS.		X101		X196 010
		X102		X196 020
		X104	0	X196 030
	X197	0		X197 000
		X20		X197 010
		X24	0	X197 020
	X198	90	0	X198 000
	90	0		X198 010
		X20		X198 020
		U134		X198 030
		X24		X198 040
		U140	0	X198 050
DESCRIBABLE N-LIST	X199	0		X199 000
		N0		X199 010
		N1		X199 020
		N2		X199 030
		N3		X199 040
		N4		X199 050
		N5		X199 060
		N6		X199 070
		N7		X199 080
		N8		X199 090
		N9	0	X199 100
KICKOFF	5	T1		T1 000
				002793

Appendix B

PROGRAM LISTING OF AN INFORMATION-ANNEXING SCHEME

The listing in this Appendix is for an annexing routine, capable of annexing to memory the contents of sets of sentences of the kinds considered in Part III.

2	A	10
2	M	100
2	N	100
2	X	200
2	Y	200
5		

5306M0. PREPARE LIST (0) FOR	M0	40H0		M000010
3073 M1. OUTPUT (0) IS VALUE		10X99		M000020
3075 (1) IS FUNCTION		J62		M000030
3076 X99= ... IS THE		J75		M000040
3077 VALUE OF ...		J50		M000050
3078PUT FUNCTION IN W0		40H0		M000060
3080		J70		M000070
3081		40H0		M000080
3083IF VALUF IS SYMBOL		J60		M000090
3084FIND IT, AND DO 90		70J7		M000095
3086		J60		M000100
3087		30H0		M000110
3091		70	90	M000120
3093		J60		M000130
3094		52H0	90	M000140
3089BRING IN FUNCTION	90	11W0		M000150
3097 AND EXIT		J6	J30	M000160
5307ASSIGN (0) AS VALUE OF THE (1)	M1	J42		M001010
3090		J21		M001020
3098		11W1		M001025
3100		40H0		
3102		M20		M001026
3103SIMPLIFY (0)		M21		M001027
3104		40H0		
3106		M20		M001028
3107		20W1		M001029
3109		J71		
3110		J71		
3111	91	11W1		M001030
3114		J83		M001040
3118		7090		M001050
3120		30H0		M001060
3122		J90		M001070
3123		J136		M001080
3124		60W2		M001090
3129		910		M001110
3130		11W1		M001140
3132		J60		M001150
3133		J68		M001160
3134		11W2		M001170
3136		20W0	91	M001180
3116	90	11W1		M001190
3139		J82		M001195
3140		910		M001210
3141		11W1		M001230
3143		J71	J32	M001240
3127ASSIGN 1W0	910	11W0		M001250
3145 AS VALUE OF J81(1W1) OF (0).		11W1		M001260
3147		J81		M001270
3148		40H0		M001280
3150IF J81(1W1)=X98		10X98		M001290
3152 PUT 1W0 AT END OF (0)		J2		M001300
3153		70J11		M001310

3155		30H0	J65	M001320
530A	M2	M10		M002010
312A		40H0		
3117		J150		
3115		J4		M002015
315A		1090		M002020
3160		J100		M002025
3161		J154		
3162		J155		
3163		10X105		M002026
3165		M4	M7	M002027
3112	90	M0		M002030
3166		M1		M002040
3167		J154		M002045
316A		J155		M002046
3169		10X105		M002050
3171		J150	J4	M002060
5309DELFT X9A TO X97 OF (0)	M3	10X97		M003010
3172		10X98	M26	M003020
5310M4, CREATE DESCRIPTION	M4	J90		M004010
3174 OF EXPRESSION (0)		J90		M04 020
3175OUTPUT (0) IS DESCRIPTION		J52		M04 030
3176		11W1		M004032
317A		10X100		M004034
3180		J65		M004036
31811W0 IS OUTPUT LIST		11W1		M04 040
31831W1 IS GENERATOR LIST		11W2		M04 050
31851W2 IS EXPRESSION		J64		M04 060
3186		11W1		M04 070
3190		1090		M04 080
3192		J100		M04 090
3193		11W0	J32	M004100
318A1W2 IS CURRENT EXPRESSION	90	40H0		M004 0110
3196		J152		
3197		60W2		
3199FIND ITS TYPE		M5		M04 120
3200ADD IT TO END OF VALUE LIST		40H0		M04 130
3202 FOR TYPE		11W0		M04 140
3204		J6		M04 150
3205		11W2		M04 160
3207		J6		M04 170
320A		J13		M04 180
3209		11W2		M04 190
3211PUT COMPONENTS ON GEN. LIST		11W1		M004200
3213		M6	J4	M004210
5311M5, FIND TYPE OF EXPRESSION (0)	M5	J0		M005010
3189		12H0		M05 015
3215OUTPUT (0) IS TYPE		J60		M05 020
3216		30H0		M05 030
3220		70	90	M005050
3222		J60		M05 050
3223		30H0		M05 060
322A		7091		M05 070
3230		10X85	0	M05 080
321A	90	30H0		M005090
3233		10X86	0	M005095
3226	91	10X45	0	M05 100
3237	92	30H0	91	M005110
5312M6, PUT COMPONENTS OF (1)	M6	J52		M006010
323A OF TYPE (2)		11W2		M06 020

32270N GENERATOR LIST (0)		10X45		M0A 030
3219NO OUTPUT		J2		M0A 040
3240		70	J32	M0A 050
3242		11W0		M006052
3244		10X100		M006054
3246		J62		M006056
3247		20W0		M006058
3249		11W2		M0A 060
3251		10X85		M0A 070
3253		J2		M0A 080
3256		7090		M0A 090
3258		11W1		M006141
3261	96	J60		M006142
3266		7092		M006143
3268		12H0		M006144
3270		11W0		M006145
3272		J6		M006146
3273		J63	96	M006147
3254	90	12W1		M0A 150
3276	91	J60		M0A 160
3278		7092		M0A 170
3280		J60		M006175
3281		12H0		M0A 180
3283		11W0		M0A 190
3285		J6		M0A 200
3286		J63	91	M006210
3264	92	30H0	J32	M0A 230
5313M7. PRINT DESCRIPTION (0)	M7	J40		M007010
3277 SAVE IT IN W0		60W0		M007020
3265 PRINTS ITS LIST OF DLS		10X86		M002030
3255		910		M007040
3288		11W0		M007050
3290 PRINT ITS LIST OF LIST		10X85		M007060
3292		910		M007070
3293		11W0		M007080
3295 PRINT ITS LIST OF SYMBOLS		10X45		M007090
3297		910	J30	M007100
3262	910	40H0		M007110
3299 PRINT ATTRIRUTE		J152		M007120
3300		J10		M007130
3304		7090		M007140
3306 PRINT LIST		J151		M007150
3302 SPACE	90	J154	J155	M007160
5316M10. READ LIST OF STATEMENT	M10	J0		M10005
3303		J154		
3301		10W25		M10010
3307 AND STORE AS LIST OF		M11		M10020
3308 LISTS (0).		10W30		M10030
3310		M11		M10040
3311SET UP OUTPUT LIST		J90		M10050
3312		J50		M10060
3313READ NEXT LINE	90	J180		M10070
3318		7091		M10080
3320SET UP STATEMENT LIST		J90		M10090
3321		J136		M10100
3322SET POINTER TO PSJ 0		11W25		M10110
3324		J124		M10120
3325SET POINTER TO FIRST SUMR		J184		M10130
3326		7091		M10135
3328		30H0		M10140

3331				
3334	92	11W30		M10150
3335MEASURE SYMR		J124		M10160
3336		J183		M10170
3338 AND SAVE IT		30H0		M10180
3340AND INPUT IT TO H0		40H0		M10100
3341PUT IT AT END OF STATEMENT LIST		J181		M10190
3342		J65		M10200
3344FIND NEXT SYMR		11W25		M10210
3345		J184		M10220
3350		30H0		M10230
3316		7093	92	M10240
3353	91	30W25		M10250
3355		30W30		M10260
3348		11W0	J30	M10270
3358	93	11W0		M10280
3359		J6		M10290
5317M11. SET UP D.T. IN (0)		J65	90	M10300
3349	M11	J50		M11010
3347		J90		M11020
3332		J124		M11030
3317		41W0		M11040
5326M20. CREATE FIND LIST		21W0	J30	M11050
3314 FROM DESCRIPTION LIST (0)	M20	J50		M020010
3360 OUTPUT (0) IS FIND LIST,		J90		M020020
3362		40H0		M020030
3366		11W0		M020040
3368		1090		M020050
3369		J100		M020060
3364		30H0	J30	M020065
3371	90	J64		M020070
5327M21. REDUCE FIND LIST (0)		40H0	0	M020080
3365 OUTPUT (0) IS REDUCED LIST	M21	J41		M021010
3374		40H0		M021020
3375		J60		M021030
3377 FIND NEXT ARGUMENT		60W0		M021040
3379		52H0		M021050
3382LOCATE NEXT DETERMINER	92	11W0		M021060
3386 IF NONE, GO TO 90		J60		M021070
3388FIND DETERMINER		7090		M021080
3390 SAVE IT		52H0		M021090
3392DETERMINE FUNCTION		60W1		M021100
3396IF NO VALUE, GO TO 91		J10		M021110
3398DELETE ARGUMENT FROM LIST		7091		M021120
3400		11W0		M021130
3401REPLACE DETERMINER WITH VALUE		J68		M021140
3384END OF LIST		61W0	92	M021150
3404	90	30H0		M021160
3394PROCESS X97 OR QUIT		30H0	J31	M021170
3407	91	11W1		M021180
3409		10X97		M021190
3410		J2		M021200
3412POINT TO PREDECESSOR OF X97		70J31		M021210
3414AND PROCESS IT		11W0		M021220
3418		M22		M021230
3420		7093		M021240
3425		61W0		M021250
3423		94	92	M021254
3426	94	J0		M021255
3428		11W0		M021256
		M24		M021260

3429DELETE X98		11W0		M021270
3431		J60	J68	M021280
3416CREATE OBJECT FOR	93	40W2		M021300
3433SURDESCRIPTION		11W0		M021310
3435COPY PART DESCRIPTION		M23		M021320
3436		20W2		M021330
3438CREATE NEW OBJECT		J90		M021340
3439		J136		M021345
3440		40H0		M021350
3442		40H0		M021360
3444		11W2		M021370
3446		J6		M021380
3447PUT IT AT HEAD OF PART DESCRIPT		J64		M021390
3448		12W0		M021400
3450		J6		M021410
3451PUT IT AT END OF ARG LIST		J65		M021420
3452		11W2		M021430
3454DESCRIBE IT		M20		M021440
3455		M0		M021450
3456		M1		M021460
3457SURSTITUTE IT FOR ARG		21W0		M021470
3459		94	J32	M021480
5328M22. IDENTIFY LIST MEMBER	M22	J0		M022010
3424		12H0		M022015
3417 WITH ARG. LOCATION (0)		J50		M022020
3415 X97 AT N(0)		M23		M022030
3395IF IDENTIFIED, OUTPUT		40H0		M022040
3385(0) IS NAME OF MEMBER.		11W0		M022050
3461IF NOT IDENTIFIED, NO		1090		M022060
3463OUTPUT. FIND LIST NOT		J100		M022070
3464		J5		M022072
3465		70	J30	M022074
3467		30H0		M022076
3469		30H0	J30	M022078
3380MUTILATED	90	60W0		M022080
3472		M25		M022090
3476		70	91	M022100
3478		40H0	J4	M022110
3474	91	51W0	J3	M022120
5329M23. COPY DESCRIPTION FROM	M23	J0		M023010
3475 X97 TO X98 OF (0)	91	J60		M023020
3460 OUTPUT (0) IS COPY		70J7		M023030
3482		12H0		M023040
3484		10X97		M023050
3486 TEST IF = X97		J2		M023060
3487		7091		M023070
3489 PUT LOC OF X97 OUT		J50		M023080
3490 CREATE NAME OF COPY		J90		M023090
3491		40H0		M023100
3493		11W0		M023110
3498DO 90 ON CELLS OF (0)		1090		M023120
3500		J100	J30	M023130
3496	90	40H0		M023140
3502IF SYMB IS X98, QUIT		10X98		M023150
3504		J2		M023160
3505		J5		M023170
3509		70	92	M023180
3511		J8	J8	M023190
3507IF NOT, ADD TO LIST	92	J65		M023200
3512		40H0	0	M023210

5330	DELETE X97 TO X98 OF (0)	M24	10X9H		M024010
3506			10X97	M26	M024020
5331	M25, TEST IF (0) FITS	M25	J51		M025010
3495	DESCRIPTION (1)	91	11W0		M025020
3515			11W1		M025030
3517	LOCATE NEXT DETERMINER		J60		M025040
3518	SAVE LOCATION		60W1		M025050
3520	FIND IT		52H0		M025060
3522			40H0		M025070
3524			10X99		M025080
3526	IS IT X99		J2		M025090
3530	IF SO, TERMINATE TEST		70	90	M025100
3532	FIND VALUE OF (0)		J10		M025110
3533	IF NONE, EXIT WITH -		70J31		M025120
3535	IF SO, SUBSTITUTE IT FOR (0)		20W0	91	M025130
3528	COMPARE VALUE WITH VALUE	90	30H0		M025140
3538	OF DESCRIPTION		11W1		M025150
3540			J60		M025160
3541			70J7		M025170
3543			52H0		M025180
3545	AND EXIT WITH SIGNAL		J2	J31	M025190
5332	M26 DELETE SECT FROM (0) TO (1)	M26	J52		M026010
3529	OF LIT (0), NO OUTPUT,		11W2		M026020
3473	TESTS THAT CN(2)=(0), IF (1)		11W1		M026030
3547	NOT ON LIST, DOES NOTHING		J77		M026040
3548			70J32		M026050
3550			11W2		M026060
3552			J60		M026070
3553			12H0		M026080
3555			11W0		M026090
3557			J2		M026100
3558			70J7		M026110
3560		90	J68		M026120
3562			11W2		M026130
3564			J60		M026140
3565			70J7		M026150
3567			12H0		M026160
3569			11W1		M026170
3571			J2		M026180
3572			7090		M026190
3574			30H0	J32	M026200
5619		5	1		X113010
5296		X113			A000010
3561		A0	0		A000013
3576			X113		A000015
3577			X99		A000020
3578			X31		A000030
3579			X20		A000040
5297			X105	0	A001010
3580		A1	0		A001013
3581			X114		A001015
3582			X99		A001020
3583			X33		A001030
3584			X25		A001040
5298			X105	0	A002010
3585		A2	0		A002013
3586			X115		A002015
3587			X116		A002017
3588			X99		A002020
			X34		

3589			X25		A002030
3590			X105	0	A002040
5407	N1	1		1	N001010
5530SDSC OF ROUTINE	X24		90	0	X24 010
3591	90		0		X24 020
3593			X32		X24 030
3594			X86		X24 040
3595			X87		X24 050
3599			91	0	X24 060
3597	91		0		X24 070
3600			X71	0	X24 080
5547NAME	X41		90	0	X41 010
3598	90		0		X41 020
3592			X32		X41 030
3601			X45	0	X41 040
5577LIST OF AFFECTED CELLS	X71		90	0	X71 010
3596	90		0		X71 020
3603			X32		X71 030
3604			X85		X71 040
3605			X91		X71 050
3606			X72	0	X71 060
5578AFFECTED CELL	X72		90	0	X72 010
3602	90		0		X72 020
3608			X32		X72 030
3609			X86		X72 045
3610			X87		X72 050
3614			91	0	X72 060
3612	91		0		X72 070
3615			X41		X72 080
3616			X75		X72 090
3617			X76	0	X72 100
5581INPUT STATE OF CELL	X75		90	0	X75 010
3613	90		0		X75 020
3607			X32		X75 030
3618			X85		X75 040
3619			X91		X75 050
3620			X45	0	X75 060
5582OUTPUT STATE OF CELL	X76		90	0	X76 010
3611	90		0		X76 020
3622			X32		X76 030
3623			X85		X76 040
3624			X91		X76 050
3625			X45	0	X76 060
5591LIST	X85				X85
5592DESCRIPTION LIST	X86				X86
5593ATTRIBUTE	X87				X87
5594OBJECT	X88				X88
5595VALUE	X89				X89
5597MEMBER TYPE	X91				X91 010
112	5		M2		
3630	3	0			
	2	3627			
	2	3637			
	2	3644			
3627	3	0			
	0	X113			
	0	X99			
	0	X31			
	0	X20			